



**NAVAL
POSTGRADUATE
SCHOOL**

MONTEREY, CALIFORNIA

THESIS

**A METHODOLOGY FOR DEVELOPING TIMING
CONSTRAINTS FOR
THE BALLISTIC MISSILE DEFENSE SYSTEM**

by

Michael H. Miklaski
Joel D. Babbitt

December 2003

Thesis Co-Advisor:	Man-Tak Shing
Thesis Co-Advisor:	James Bret Michael

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE December 2003	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: A Methodology for Developing Timing Constraints for the Ballistic Missile Defense System			5. FUNDING NUMBERS	
6. AUTHOR(S) Michael H. Miklaski and Joel D. Babbitt				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			12b. DISTRIBUTION CODE A	
13. ABSTRACT (maximum 200 words) The Department of Defense (DoD) is developing a Ballistic Missile Defense System (BMDS) based on a layered defense that employs complementary sensors, weapons and C2 elements integrated by software into a system-of-systems to engage and destroy threat ballistic missiles through all phases of its flight. Inherent to the ultimate success of the BMDS will be the timely execution of the kill chain process against threat ballistic missiles. In this thesis we will apply the Unified Software Development Process, utilizing the BMDS as a case study, to investigate a means to identify and validate timing behaviors and constraints of system-of-systems. In particular, we will examine the information exchange needed for processors to share, collaborate, fuse, and distribute sensor information in a distributed sensor network and utilize modeling and simulation to provide insight into the timing aspects of interactions among subsystems comprising a system-of-system. The case study will involve deriving and documenting system and software requirements, developing a test-ready model for representing the timing requirements, and then validating this model through the use of an OMNET++ simulation. The simulation will then be used to provide feedback to further refine the system requirements and the functional specifications of the subsystems.				
14. SUBJECT TERMS Software Engineering, System-of-Systems, Ballistic Missile Defense System (BMDS), Sensor Fusion, Collaborative Fusion, Modeling, Simulation, OMNeT++, UML-RT, Real-Time Constraints, Software Requirements, Kill Chain, Timing Requirements, Unified Software Development Process.			15. NUMBER OF PAGES 309	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

**A METHODOLOGY FOR DEVELOPING TIMING CONSTRAINTS FOR
THE BALLISTIC MISSILE DEFENSE SYSTEM**

Michael H. Miklaski
Commander, United States Navy
B.S., National University, 1987

Submitted in partial fulfillment of the
requirements for the degrees of

**MASTER OF SCIENCE IN SYSTEMS TECHNOLOGY
and
MASTER OF SCIENCE IN SOFTWARE ENGINEERING**

from the

**NAVAL POSTGRADUATE SCHOOL
DECEMBER 2003**

Joel D. Babbitt
Captain, United States Army
B.S., Brigham Young University, 1995

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
MARCH 2004**

Authors: Michael H. Miklaski

Joel D. Babbitt

Approved by: Man-Tak Shing
Thesis Co-Advisor

James Bret Michael
Thesis Co-Advisor

Dan C. Boger
Dean, Department of Information Sciences

Peter J. Denning
Dean, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The Department of Defense (DoD) is developing a Ballistic Missile Defense System (BMDS) based on a layered defense that employs complementary sensors, weapons and C2 elements integrated by software into a system-of-systems to engage and destroy threat ballistic missiles through all phases of its flight. Inherent to the ultimate success of the BMDS will be the timely execution of the kill chain process against threat ballistic missiles.

In this thesis we will apply the Unified Software Development Process, utilizing the BMDS as a case study, to investigate a means to identify and validate timing behaviors and constraints of system-of-systems. In particular, we will examine the information exchange needed for processors to share, collaborate, fuse, and distribute sensor information in a distributed sensor network and utilize modeling and simulation to provide insight into the timing aspects of interactions among subsystems comprising a system-of-system. The case study will involve deriving and documenting system and software requirements, developing a test-ready model for representing the timing requirements, and then validating this model through the use of an OMNET++ simulation. The simulation will then be used to provide feedback to further refine the system requirements and the functional specifications of the subsystems.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
II.	BMD OVERVIEW.....	7
A.	INTRODUCTION.....	7
B.	A BRIEF HISTORY OF BALLISTIC MISSILE DEFENSE.....	7
C.	DIRECTION OF BMD.....	10
D.	LEGACY SYSTEMS.....	12
E.	SYSTEMS OF SYSTEM APPROACH.....	16
III.	BMDS OPERATING ENVIRONMENT.....	19
A.	INTRODUCTION.....	19
B.	THE KILL CHAIN.....	21
1.	Surveillance.....	22
2.	Detection.....	22
3.	Tracking.....	23
4.	Identification.....	24
5.	Target/Engage.....	25
6.	Assess.....	26
C.	PHASES OF FLIGHT.....	27
1.	Boost Phase.....	27
2.	Midcourse.....	29
3.	Terminal.....	29
D.	BMDS COMPONENTS.....	30
1.	Sensors.....	31
2.	Weapons.....	35
3.	Command, Control, Battle Management, and Communications (C2BMC).....	37
IV.	BMDS REQUIREMENTS SPECIFICATION.....	39
A.	INTRODUCTION.....	39
B.	VISION AND SOFTWARE REQUIREMENT SPECIFICATION (SRS) DOCUMENTS.....	40
C.	DESCRIPTION OF BMDS ARCHITECTURE.....	42
D.	USE CASES.....	49
1.	Use Case 1: Detect Potential Threat Ballistic Missile.....	51
2.	Use Case 1.1: Generate and Transmit a Local Track.....	54
3.	Use Case 2: Cooperatively Track and Classify Threat Ballistic Missiles.....	56
4.	Use Case 3: Cooperative Weapons Assignment.....	59
5.	Use Case 4: Engage Targets.....	62
6.	Use Case 5: Assess Kill.....	64
E.	CLASS DIAGRAM.....	66
F.	SYSTEM SEQUENCE DIAGRAMS (SSD).....	66

V.	BMDS MODEL.....	69
A.	INTRODUCTION.....	69
B.	CONTEXT.....	73
C.	ASSESSMENT.....	74
VI.	BMDS OMNET ++ SENSOR FUSION PROCESSOR (SFP) SIMULATION	77
VII.	DISCUSION OF RESULTS	83
VIII.	CONCLUSION.....	91
A.	SUMMARY.....	91
B.	RECOMMENDATIONS.....	91
APPENDIX A.	GLOSSARY.....	93
APPENDIX B.	VISION DOCUMENT	113
APPENDIX C.	SRS DOCUMENT	123
APPENDIX D.	SYSTEM SEQUENCE DIAGRAMS (SSD).....	133
A.	SSD FOR HIGH-LEVEL BMDS USE CASE.....	133
B.	SSD FOR USE CASE 1 & 1.1	133
C.	SSD FOR USE CASE 2	134
D.	SSD FOR USE CASE 3	134
E.	SSD FOR USE CASE 4	135
F.	SSD FOR USE CASE 5	135
APPENDIX E.	UML-RT MODELS	143
A.	SENSOR.....	143
B.	SENSOR CONTROLLING AUTHORITY	144
C.	COMPETENT AUTHORITY	145
D.	SENSOR FUSION PROCESSOR (SFP)	146
E.	SFP'S SENSOR INTERFACE CAPSULE.....	147
F.	SFP'S TRACK FUSING CAPSULE	149
G.	SFP'S COLLABORATIVE FUSING CAPSULE.....	151
H.	SFP'S TRACK LIST CAPSULE	153
I.	SFP'S SENSOR NET INTERFACE CAPSULE.....	155
J.	SENSOR NET.....	156
K.	SENSOR NET'S SFP INTERFACE CAPSULE.....	158
L.	SENSOR NET'S TRACK FILTER CAPSULE.....	160
M.	SENSOR NET'S CUEING CAPSULE	162
N.	SENSOR NET'S TRACK REGISTRY CAPSULE.....	164
O.	SENSOR NET'S TRACK SERVER CAPSULE.....	165
P.	SENSOR NET'S PEER/HIGHER INTERFACE CAPSULE	167
Q.	SENSOR NET'S WEAPONS PLATFORM INTERFACE CAPSULE .	168
R.	WEAPONS PLATFORM.....	170
S.	BMC2.....	171
T.	WEAPON.....	172
U.	WEAPON NET.....	173
APPENDIX F.	SIMULATION CODE	175

A.	SFP SIMULATION CODE.	175
B.	SENSOR NET SIMULATION.	207
APPENDIX G.	SIMULATION DATA	269
LIST OF REFERENCES.	285
BIBLIOGRAPHY	287
INITIAL DISTRIBUTION LIST	291

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1.	BMDS Kill Chain Function.....	21
Figure 2.	An Integrated, Layered Defense Against Missiles of All Ranges	27
Figure 3.	BMDS Sensor Diagram.....	31
Figure 4.	BMDS Interceptor Diagram.....	35
Figure 5.	Process of BMDS development	39
Figure 6.	Distributed C2BMC Architecture.....	42
Figure 7.	High-Level BMDS Use Case.....	50
Figure 8.	Use Case 1 Diagram.....	51
Figure 9.	Use Case 2 Diagram.....	56
Figure 10.	Use Case 3 Diagram.....	59
Figure 11.	Use Case 4 Diagram.....	62
Figure 12.	Use Case 5 Diagram.....	64
Figure 13.	BMDS Class Diagram (After Ref. Conceptual Framework Approach for Systems-of-Systems Software Development).....	67
Figure 14.	The Miracle	69
Figure 15.	OMNet++ BMDS SFP Simulation.....	77
Figure 16.	SFP Track List Capsule Redesign	89
Figure 17.	SSD for High-Level Use Case	136
Figure 18.	SSD for Use Case 1 & 1.1.....	137
Figure 19.	SSD for Use Case 2.....	138
Figure 20.	SSD for Use Case 3.....	139
Figure 21.	SSD for Use Case 4.....	140
Figure 22.	SSD for Use Case 5.....	141
Figure 23.	Sensor UML-RT Diagram.....	143
Figure 24.	Sensor Controlling Authority UML-RT Diagram...	144
Figure 25.	Competent Authority UML-RT Diagram	145
Figure 26.	Sensor Fusion Processor UML-RT Diagram	146
Figure 27.	Sensor Interface Capsule UML-RT Diagram	147
Figure 28.	Track Fusing Capsule UML-RT Diagram	149
Figure 29.	Collaborative Fusing Capsule UML-RT Diagram...	151
Figure 30.	Track Capsule List UML-RT Diagram	153
Figure 31.	Sensor Net Interface Capsule UML-RT Diagram...	155
Figure 32.	Sensor Net UML-RT Diagram.....	156
Figure 33.	SFP Interface Capsule UML-RT Diagram	158
Figure 34.	Track Filter Capsule UML-RT Diagram	160
Figure 35.	Cueing Capsule UML-RT Diagram.....	162
Figure 36.	Track Registry Capsule UML-RT Diagram	164
Figure 37.	Track Server Capsule UML-RT Diagram	165
Figure 38.	Peer/Higher Interface Capsule UML-RT Diagram..	167
Figure 39.	Weapons Platform Interface Capsule UML-RT Diagram	168
Figure 40.	Weapons Platform UML-RT Diagram	170

Figure 41.	BMC2 UML-RT Diagram.....	171
Figure 42.	Weapon UML-RT Diagram.....	172
Figure 43.	Weapon Net UML-RT Diagram.....	173
Figure 44.	Varying Data Rates and Track Message Sizes....	271
Figure 45.	Ground-based Radar Update Delay	272
Figure 46.	Space-Based IR Update Delay	274
Figure 47.	Varying Number of Ground-based Radar Sensors..	275
Figure 48.	Varying Number of Space-based IR Sensors.....	277
Figure 49.	Collaborative Fusion Requests	278
Figure 50.	Module Processing Time.....	280
Figure 51.	Track List Access Time.....	281
Figure 52.	Time to Perform Track Fusion	283

LIST OF TABLES

Table 1.	Varying Data Rates	269
Table 2.	Varying Track Message Sizes	270
Table 3.	Ground-based Radar Update Delay.....	273
Table 4.	Space-Based IR Update Delay	273
Table 5.	Varying Number of Ground-based Radar Sensors	276
Table 6.	Varying Number of Space-based IR Sensors	276
Table 7.	Collaborative Fusion Requests	279
Table 8.	Module Processing Time	279
Table 9.	Track List Access Time	282
Table 10.	Time to Perform Track Fusion.....	282
Table 11.	Master Track List Broadcast Times	284
Table 12.	Capsule Data Rate.....	284

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

CDR Michael Miklaski

I would like to express my undying love and gratitude to my wife, Jane, who has always been stood by me in all my endeavors providing support and a swift kick when necessary, and to my children Jessica and Matthew, whose mere presence is a constant reminder that I must be continuously learning. Also, I will always cherish the not so subtle reminders that if they had to do their homework I had to do mine as well and the periodic comparison of report cards to see who had the better grades (it wasn't me). I also wish to express a heartfelt "Thanks" to all the Software Engineering and C4I professors, and in particular Professors Michael and Shing, for their extreme patience in attempting to impart their knowledge to an old dog trying to learn new tricks.

CPT Joel Babbitt

My deepest thanks to God for everything, including my meager modeling and programming skills. Thanks ever so much to my wife, Kristen, for putting up with my work ethic, for listening to much talk laced with many cryptic terms, and for standing by me through this entire process. Thanks to my boys, Xander and James, for giving up their daddy 'to the thesis monster'. Thanks to Professors Shing and Michael for the MANY hours and constant dedication to seeing our thesis through, and to Professor Bill Ray as well for his programming assistance. Finally, thanks to the many professors and fellow students here at Naval Postgraduate School from whom I have learned so much.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

I keep six honest serving-men
(They taught me all I knew);
Their names are What and Why and When
And How and Where and Who.
I send them over land and sea,
I send them east and west;
But after they have worked for me,
I give them all a rest.
I let them rest from nine till five,
For I am busy then,
As well as breakfast, lunch and tea,
For they are hungry men.
But different folk have different views.
I know a person small --
She keeps ten million serving-men,
Who get no rest at all!
She sends 'em abroad on her own affairs,
From the second she opens her eyes --
One million Hows, two million Wheres,
And seven million Whys!

Rudyard Kipling, The Elephant's Child (1902)

The primary goal of our thesis is to continue the development, refinement, and documentation of the high-level requirement specification, baseline architecture, and real-time model of a notional Ballistic Missile Defense System (BMDS) that was started in earlier work.¹ In particular, our focus is to try to determine what the potential timing constraints are on the BMDS we are

¹ Dale Scott Caffall, "Conceptual Framework Approach for System-of-Systems Software Developments" (M.S. Thesis, Naval Postgraduate School, Mar. 2003)

developing. We then will create a high-level simulation of the BMDS that can validate those derived requirements and assess the timing constraints of the BMDS. This simulation will also be designed such that it can be reused for further research on the subject.

We intend to utilize establish software engineering practices that have been the bedrock of our graduate education to achieve these stated goals, and in particular we will utilize the Unified Software Developmental Process (USDP) to develop the BMDS. The USDP is a use case driven incremental and iterative process consisting of five core workflows (requirements, analysis, design, implementation, and testing) and four phases (inception, elaboration, construction, and transition).² In being iterative and incremental we can break the project down into smaller parts to analyze, design, implement, and test, and to make any necessary changes.

As has been observed in some software projects, those practitioners who are doing the developing may have only a limited insight, if any at all, into the product being constructed. It is paramount that the developers gain a fundamental understanding as part of the development process. One needs to establish a solid foundation of understanding of "what problem it was we were trying to solve and why we are trying to solve it"³ by putting our "honest men" to work and researching the problem.

Armed with a rudimentary knowledge of Ballistic Missile Defense (BMD) and the systems that comprise it, a host of software engineering classes, a development

² Simon Bennett, John Skelton, and Ken Lunn, *Schaum's Outline UML*, McGraw-Hill, London, 2001, pp 20-21.

³ Professor Richard Riehle, Naval Postgraduate School, Jan. 2003.

process, and a profound quote from Kipling passed on to us; we embark upon our journey of discovery.

The first issue at hand is the need to establish who is driving the need for a BMDS, in essence who is the customer, and why such a system is required: the answer comprises Chapter II, in addition to a history of BMD, answering why current systems cannot fulfill the future needs of a BMDS, and the approach we intend take to achieve the goals of our thesis.

Next we looked at what methodology the BMDS is to be implemented and integrated, and how that system is tactically envisioned to be employed in the prosecution of threat ballistic missiles. We use a course-grain model of BMDS in order to reason about what specific sensors, weapons, battle manager, and command and control systems are intended to comprise a BMDS system-of-systems. This is done to gain a fundamental understanding into the required functionality and overall system behavior. All of which serve as the basic template as we continue development of the BMDS. In essence, Chapter III becomes the requirements elicitation phase of the software requirements specification process.

The process of developing the requirements specification and related documentation is the focus of Chapter IV. We start by creating the vision and Software Requirement Specification documents and describing the BMDS architecture based on the information derived through the requirements elicitation phase. This is followed by the process of specifying the system requirements by utilizing use cases to identify the who, what, and how of the BMDS behavior. The use cases are then realized via collaborations consisting of a static class diagram and

dynamic system sequence diagrams. From the use cases, the BMDS class diagram from previous work is further refined and expanded annotating the new derived classes, attributes, and messages that occur between those classes to statically describe the BMDS. System sequence diagrams (SSD) are developed to show dynamic behavior of the BMDS through the necessary communication between the classes and objects of the BMDS via messages and the timeline in which those messages must occur in order to realize successful system operation in the prosecution of threat ballistic missiles. From these artifacts we then develop a real-time, high-level model that can start to identify the actual timing constraints that will be imposed upon the notional BMDS.

To make the transition from understanding the requirements of the BMDS to the design and implementation of the simulation we utilize a real-time variation of the Unified Modeling Language, commonly referred to as UML-RT. UML-RT is designed specifically to model the software architectures of complex, event-driven, and distributed real-time systems to ensure that the essential structural and behavioral framework upon which all other aspects of the system depend are designed correctly and can accommodate changes over time.⁴ In developing the BMDS model with UML-RT we will gain a better understanding of the system-of-systems we are developing through visualization, behavior and structure specification, decision documentation, as well as creating a construction template from which we can start to build a prototype of the system to validate the derived requirements.

⁴ Bran Selic and Jim Rumbaugh, *Using UML for Modeling Complex Real Time Systems*, April 1998, pp 2-3.

A simulation model is defined as:

An algorithmic representation of a system, reflecting system structure and behavior, that explicitly recognizes the passage of time, hence providing a means of analyzing the behavior of the system over time.⁵

In order to perform a system analysis of the BMDS we will develop a OMNeT++ discrete event simulation of the Sensor Fusion Processor as a model for simulating the entire system, using the UML-RT model as a template for incorporating system requirements based on the documented artifacts. The simulation is used to determine whether the requirements have been achieved, that the system operates within acceptable parameters, and to discover any other possible timing considerations and constraints. The simulation is designed to allow for further research and development as the BMDS evolves.

⁵ Hassan Gomaa, *Designing Concurrent, Distributed, and Real-Time Applications with UML*, Addison-Wesley, 2000, p 752.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BMD OVERVIEW

A. INTRODUCTION

The purpose of this chapter is to provide the reader a background and insight into those factors that have driven the need for the development of a BMD through a quick look at the history of BMD and the current decisions being made by the national leadership that have and will continue to affect the development of the BMDS, in essence answering the why and who. This is followed with a brief explanation of why legacy systems cannot fill the bill for future BMDS growth and why development of a new BMDS following a system-of-systems approach is necessary.

B. A BRIEF HISTORY OF BALLISTIC MISSILE DEFENSE

On September 8th 1944, the quest for an anti-ballistic missile defense system began in earnest to counter German V-2 rockets launched against civilian targets in France and England. Initially, the only means of defense against these weapons of terror was to either locate and destroy the launch sites or occupy sufficient territory as to place the missiles out of range of civilian population centers. However, the Germans simply moved these weapons to more secure areas and continued to deploy them at targets within the operational range of the missile. By the end of the war it was determined that over 3000 V-2's had been launched with the majority of them targeted at London and Antwerp. While militarily these weapons had little impact, the political and psychological effects were significant.

Fortunately for the allies, the V-2 missiles were expensive to build, its guidance system was not highly accurate, the missile itself was unreliable, and the weapon

was introduced too late in the conflict to significantly affect the outcome of the war. However, the V2 was a harbinger of future warfare. With advances in missile technology, weapons development to include all forms of weapons of mass destruction (WMD), and more efficient and less costly production of ballistic missiles, they became quite an attractive means of bolstering the military capability of a country without bankrupting the economy.

During the Cold War, the prospect of nuclear annihilation via the exchange of Intercontinental Ballistic Missiles (ICBM) between the U.S. and Soviet Union led to the implementation of numerous arms limitations treaties; the treaties placed constraints on the use of these weapons and the platforms with which to deliver them. These treaties themselves became a means of providing BMD in that while technology was advancing for both sides it was not mature enough to develop a comprehensive, integrated system that could counter such a threat. Those systems developed in the Cold War could only track the incoming warheads and attempt to destroy the reentry vehicles (RV) in endoatmospheric reentry phase with a nuclear defensive missile, such as Nike/Zeus in the case of the U.S., while conducting a nuclear retaliatory strike to prevent further launches. The concept of mutual assured destruction (MAD) served as much of a deterrent than any fielded defensive system during the Cold War.

As technology advanced, particularly with lasers and computers, the Reagan Administration pursued in earnest the development a space-based national BMDS known as the Strategic Defense Initiative (SDI), commonly referred to as the "Star Wars" program. However, the ABM treaty signed in 1972, while allowing research, precluded actual testing or

deployment of such a system. These restrictions helped to eliminate the fear that one side could possibly gain the advantage of protecting themselves thus rendering the adversary's weapons impotent, and with that distinct advantage possibly emboldening them to preemptively launch a first strike safely in the knowledge that they could repel a counter ballistic missile strike.

With the fall of the Soviet Union, the potential for the use of ballistic missiles has actually increased. This is primarily due to proliferation of Theater Ballistic Missiles (TBM) to Soviet client states during the Cold War, the selling of technology afterward by former Soviet states, and the fact that the control leveraged over those client states by Russia to keep them in line no longer existed. This is evidenced by the proliferation of TBM in Third World countries such as Iraq and North Korea that possess Soviet-made missiles and using the technologies acquired to develop homegrown TBM's such as the No Dong I, Taepo Dong I/II, and all variants of the SCUD, which currently threaten the U.S. and her allies.

U.S. forces' first real exposure to a TBM threat occurred during Operation Desert Storm in which military history was made with the first successful intercept of a SCUD by a Patriot missile. On commencement of the Coalition air war, Iraq commenced SCUD attacks against targets in Saudi Arabia and Israel. While tactically insignificant, the eighty-eight SCUD missiles that were launched in the resulting terror campaign nearly drew Israel into the conflict, which could have both unraveled the Coalition and resulted in the loss of support from Arab nations. It was following this campaign that a significant amount of DoD focus was directed at the countering the

ballistic missile threat, and lead to the establishment of the Ballistic Missile Defense Organization (BMDO), later MDA, and the Joint Theater Air and Missile Defense Organization (JTAMDO), who have been mandated to develop a BMDS.

C. DIRECTION OF BMD

On December 13, 2001, President Bush announced to Russia and the world that the United States, after reeling from the devastating terrorist attacks of September 11 and facing new threats since the end of the Cold War, in particular rogue states and terrorist groups possessing WMD and ballistic missiles with which to deliver them, was serving the required six months prior notice necessary to pull out of the Anti-Ballistic Missile (ABM) Treaty of 1972.⁶ This major decision has had a profound and dramatic impact on the National Security Strategy of the U.S. and has led to the direct and stated goal of developing a BMDS. By pulling out of the ABM Treaty, the President has now made it possible to fully develop and test BMD systems that were previously restricted to research only and if capable, to deploy those systems as desired.

In order to facilitate the development of a BMDS, just two weeks after the President's announcement, Secretary of Defense Rumsfeld, in a memorandum dated 2 January 2002, announced the restructuring of the entire National Missile Defense (NMD) Program placing all programs under the Missile Defense Agency (MDA): MDA, reports directly to the Under Secretary of Defense for Acquisition, Technology and Logistics (USD AT&L), and was provided guidance for the

⁶ <http://www.whitehouse.gov>, Press release 13 Dec. 2001.

development and employment of an integrated and layered BMDS that will be able to

detect, track, intercept and defeat ballistic missiles in all phases of their flight (i.e., boost, midcourse, and terminal) against all ranges of threats.⁷

Additionally, the memo streamlined the acquisition process for all related BMD systems by removing them from constraining government instructions and directives, allowing MDA to pursue a capabilities-based approach toward BMD systems research, development, test and evaluation.

In an effort to make BMD a reality, President Bush announced to the nation and the world,

I made a commitment to transform America's national security strategy and defense capabilities to meet the threats of the 21st century... I have directed the Secretary of Defense to proceed with fielding an initial set of missile defense capabilities. We plan to operate these initial capabilities in 2004 and 2005, and they will include ground-based interceptors, sea-based interceptors, additional Patriot units and sensors based on land, sea, and in space.⁸

By committing to field systems in an incremental fashion as those systems are developed, the intent is to deploy an initial system, which can be continuously modified and made more robust over time.

In order to achieve these stated goals and critical to the success of the implementation -- and both at the heart of the BMD System and the evolving BMD Strategy -- is the development and future employment of the Command, Control, Battle Management and Communication (C2BMC) System. The

⁷ Office of the Secretary of Defense, SecDef Memo dated 2 Jan. 2002.

⁸ <http://www.whitehouse.gov>, Press release 17 Dec. 2002.

ability to detect, track, identify, and target threat ballistic missiles in all phases of their flight and provide weapons systems the accurate and timely information necessary to consummate an intercept is the primary goal of BMD.

D. LEGACY SYSTEMS

In this thesis we lay aside any restrictions imposed by legacy systems, as the cost of upgrading them is prohibitive and the restrictions they bring with them without a complete upgrade would make the President's long term goal of defending the nation from ballistic missile threats untenable. As such, we have striven to incorporate the latest component-based system design methodologies to provide for ease of system decomposition and evolution of the system as threats, doctrine, and technology change. However, these systems initially will need to be utilized until a more robust and responsive system can be implemented. Therefore, these legacy systems need to be discussed; particularly with reference as to why they cannot be improved upon to fulfill the future needs of the envisioned BMDS.

Prior to Desert Storm, with the primary threat being ICBMs stationed in the former Soviet Union, the BMD System consisted of large, fixed radar sites in the northern latitudes, and Defense Support Program (DSP) satellites in geosynchronous orbits, scanning the predicted avenues of approach of possible ICBM attacks and nuclear-tipped defensive missiles poised to intercept incoming ICBM's. These systems were considered national assets and strategic in nature, reporting directly to NORAD HQ at Cheyenne

Mountain, Wyoming, which in turn reported to the National Command Authority (NCA) in Washington D.C.

BMD weapons-system development prior to Desert Storm was not predicated on the possibility of having to conduct integrated BMD at a tactical level, and therefore was not optimized to perform that mission. In fact, most weapons systems were developed in a "stovepipe" fashion that necessitated significant modifications to existing software systems or unique network designs to allow interoperability within an established command and control (C2) infrastructure. Each of the services have developed data links that provided connectivity between their particular units such as the Ground Based Data Link (GBDL) and Army Tactical Data Link One (ATDL-1) for Army and Marine ground units, the PATRIOT Digital Information Link (PADIL) specifically designed for the PATRIOT system, and the Navy's Link-4A (TADIL-C) for two-way interceptor air control. None of these data links have a direct means to integrate with one another and must all be translated into another C2 data link, such as Link-11 or Link-16, in order to transmit and receive time-critical information and achieve connectivity and interoperability.

Proliferation of smaller theater and tactical ballistic missiles to Third World countries and the advent of Desert Storm necessitated a change in the design of weapons and C2 systems. During Desert Storm, the Coalition forces developed and deployed a defensive system to counter Iraqi SCUD missiles that are not much more sophisticated than the German V-2 rocket. The theater-level air-defense system consisted of sensor and weapon systems tied together through a variety of previously mentioned legacy data link systems in order to develop a coherent air picture. The

complexity of the architecture and the limitations of those older data link systems required the coalition to develop custom patches to attain a comprehensive defense. For instance, initially Link-11 was implemented as a single data link with all possible units participating. However, as units joined the network and started to enter tracks the sheer volume of all the track data saturated many of the participants who had relatively limited track file capacity and prevented many critical contacts from being presented in a timely fashion. To address this problem, multiple Link-11 nets were established with gateway units filtering data as necessary to preclude saturation.

While the aforementioned "fixes" worked to a limited degree, latency due to the numerous translations among disparate data links and communications systems, lack of complete connectivity, and differing navigational data precluded attainment of an accurate, near real-time C2 environment. This was evident by the minimum response time available to the Patriot batteries in response to SCUD launches during Operation Desert Storm despite early launch detection by Defense Support Program (DSP) satellites.

Since the conclusion of Desert Storm, systems such as the Cooperative Engagement Capability (CEC) and the Joint Tactical Information Distribution System (JTIDS) have been introduced to increase the throughput of data, enhance the overall situational awareness of participating units, and in the case of CEC' providing a composite track picture consisting of radar-parametric and identification data so that units outside the detection range of a target can actually launch on remote. These systems have been utilized in operational and test and evaluation exercises to develop an effective, recognizable air picture and are

anticipated to migrate into the arena of BMD. In fact, Link 16/JTIDS has been identified as the initial C3 system to integrate the BMDS with the potential to provide a Single Integrated Air Picture (SIAP) for BMDS through CEC sensor fusion. However, a recent study conducted by the Naval Studies Board of the National Research Council found that both of these systems would be inadequate in the long run for a BMDS.

As previously described, the JTIDS/Link 16 approach is a bandwidth limited, rapidly obsolescing technology that will impede future operational flexibility. There are a variety of planned improvements that may make it somewhat more effective, and these should be continued as planned. However, at each stage, the Navy should evaluate the utility and cost of the improvements against the evolving capability provided by the Internet technology prototyping. The goal should be to use JTIDS/Link 16 when nothing better is available but to wean the BMC3 system from depending on it.

CEC is an excellent implementation of the philosophical approach advocated by the committee in that it seeks to accommodate distributed sensors. It provides the basis for the current self-defense capabilities and gives the Navy some area defense capability. It is, however, a closed-loop system that will not provide the long-term capabilities needed for a more complete TMD BMC3.⁹

Therefore a new and more modern approach, using the latest system and software engineering methodologies available to integrate all of the subsystems that will comprise the BMDS, needs to be undertaken to develop the software for the follow-on BMC3 system. Inevitably, "Software, not hardware, will determine the ultimate

⁹ Naval Studies Board National Research Council, *Naval Forces Capability for Theater Missile Defense*, National Academies Press 2001, p 162.

functionality of the system and the success of the system in the end user's hands..."¹⁰

E. SYSTEMS OF SYSTEM APPROACH

Dealing with systems of complexity requires nontrivial approaches, and a system of subsystems is a means to this end. Surely the alternatives are worse, as we would end up with incredibly complex systems that no one could possibly understand, with indeterminate behavior, and design based on shared functionality, poor partitioning, and threaded code in such a way as could never be unraveled. ... And what happens if we don't do a good job of systems engineering? The system will become brittle and will resist change because of the weight of the requirements assets will "bind" us to the implementation. Our subsystem requirements have taken control of our design flexibility, and a change in one will have a ripple effect in other subsystems. These are the "stovepipes" systems of legend, and such systems resist change. In their interfaces, the problems may be worse. If the interfaces are not properly specified, the system will be fragile and will not be able to evolve to meet changing needs without the wholesale replacement of interfaces and the entire subsystems that were based on them.¹¹

As previously shown, current C3 systems, while providing an initial capability to fulfill President Bush's mandate, will not be able to grow to meet the projected demands nor take advantage of advances of current and future commercial technology. The reality is that even as JTIDS is being fielded, it has been in development for over thirty years and has already become a legacy stovepipe system with minimal room for future growth. One of the major problems with JTIDS is that it requires significant

¹⁰ Dean Leffingwell, Don Widrig, *Managing Software Requirements*, Addison-Wesley, 2000, p 63.

¹¹ Ibid, p 65.

lead-time of about one to two weeks and foreknowledge of the participating platforms in order to develop the network with which to integrate those systems and distribute the software to the units. CEC, which is outstanding at developing a local SIAP, requires very specific equipment and software to implement a sensor fusion network and does not have the capacity to provide the complete set of data that a tactical data link does.

In BMD scenarios it is not envisioned that a sufficient amount of time will be available to configure a JTIDS network rapidly enough to meet the threat or to include whatever units that are available to operate together in a cohesive manner to affect a proper defense. Also, as systems are modified or developed over time they must be able to participate in the network with a minimum of overhead and impact, that is be plug-and-play, and neither of these systems will be able to offer this capability even after significant planned system evolution.

Continuing the process of systems-of-systems development utilizing the BMDS as a case study¹², we intend to extend the study to the next level of realization by utilizing established software engineering requirements specification practices to further define the conceptual system of systems and develop a network simulation based on those findings in an attempt to capture the high level timing constraints imposed upon the system. These will later serve as a vehicle for continued study and refinement of the conceptual system of systems and provide an initial

¹² Caffall, D. S. and Michael, J. B. "A new paradigm for requirements specification and analysis of system-of-systems". Wirsing, M., Balsamo, S., and Knapp, A., eds., *Lecture Notes in Computer Science: Proc Monterey Workshop 2002: Radical Innovations of Software and Systems Engin. in the Future*, Berlin: Springer-Verlag, 2003.

simulation of the network that can be expanded for utilization in further research.

III. BMDS OPERATING ENVIRONMENT

A. INTRODUCTION

The Standish Group conducted a survey in 1994 to determine what were the most common factors associated with software projects that met with significant problems. One of the three most commonly identified faults was incomplete requirements specification.¹³ As part of the requirements elicitation process for this project we need to determine and understand under what paradigm or methodology the BMDS is to be developed, what sub-systems are going to comprise the BMDS, and how they are intended to operate together as a system-of-systems. Once this information has been determined the requirements specification process can be documented and software design can commence which in turn leads to identifying the timing constraints of the BMDS. The purpose of this chapter is to provide the information that is critical for identifying capabilities, functional requirements, and non-functional requirements.

MDA has set clear expectations and guidelines under which the BMDS Battle Manager (BM) is to be developed.

The BMDS BM will substantially enhance BMDS effectiveness beyond that achievable by stand-alone systems. The BM component integrates kill chain functions (surveillance, detect / track / classify, engage and assess) across the layered defenses (boost, mid-course, terminal, and external sensors (Space Based Infrared System Low - SBIRS Low)) and evolves with the BMDS elements. Initially, BM will deliver the hardware/software (HW/SW) necessary to provide the means for executing pre-planned responses by integrating available information to provide the user with increased automation capability and ability to integrate information from increasingly diverse

¹³ The Standish Group, *Charting the Seas of Information Technology*, 1994.

resources. BM will eventually provide a highly flexible and configurable framework for real time, adaptive coordination of missile defense assets, while also supporting the incorporation of new elements.¹⁴

Each one of the functional areas in the kill chain and phases of flight for a ballistic missile places unique requirements on the overarching BMD system-of-systems, as well as potential trade-offs among the non-functional requirements (e.g., timing vs. safety) that need to be addressed; each requirement has an impact on timing constraints that need to be identified and evaluated. Lastly, the types of systems that will comprise the BMDS, sensors, weapons, and C2, will need to be evaluated to determine what each of the timing requirements are in order to ensure that when they are integrated as a system-of-systems will operate effectively in the prosecution of a ballistic missile threat.

¹⁴ MDA Exhibit R-2 RDT&E Budget Item Justification (PE 0603889C), Feb. 2003.

Assign Weapon

- Maintain SA
 - Receive and Display Track Data
 - Synchronize to Ongoing Tactical Situation
- Evaluate Threat
 - Weigh Data
 - Threat Evaluation
 - Prioritize Threats
- Pair Weapon to Target
 - Evaluate Impact Point Prediction Against DAL
 - Evaluate Weapon Kinematics Constraints
 - Evaluate Weapon Sensor/Target Combinations
 - Determine Optimum Engagement Sequence
- Direct Execution
 - Assign Weapon/Sensor/Target
 - Assign Backup

Track

- Associate Track/Correlation
 - Generate Tracks
 - Correlate Local & Remote Tracks
 - Process non-organic sensor data
 - Maintain Track Files
- Discriminate
 - Apply features recognition
 - Trajectories
 - Countermeasures
 - Decoys
 - Tank/Booster
 - RV
- Identify
 - Compare Profiles
 - Trajectories
 - Phenomenology
 - RCS
- Classify
 - Points
 - Ellipse
- Develop IPP/LPE
 - Position
 - Velocity
 - Covariance
 - Sigma
 - Missile Type
 - RV Type
- Update Track Amplifying Data

Engage

- Prepare for Launch
 - Ready Interceptor
 - Ready Launcher
 - Compute firing solution
- Launch
 - TTL
 - LTL
- Flyout
 - FTU
 - Divert
- Handover
 - Activate all KV functions
 - Begin search
- Conduct Endgame
 - Fire thrusters
 - Maneuver to target

Detect

- Planned Search
 - Conduct Active Search
 - Conduct Passive Search
- Cued Search
 - Receive Tracks from other source
 - Compute search volume
- Detect
 - Process sensor data
 - Initial filtering
- Acquire
 - Develop initial track file

Assess Kill

- Collect Data
- Formulate Assessment
 - Perform discrimination
 - Compare Features
- Release Results

Feedback Loops:

- Track → Evaluate Threat → Assign Weapon
- Engage → Assess Kill → Detect
- Assess Kill → Assign Weapon
- Assess Kill → Track
- Assess Kill → Engage

1 v18

The kill chain template (Figure 1) is utilized by virtually every weapons system to describe its functionality and to determine its effectiveness. As mentioned in the previous quote, it is also a required performance function of the BMDS and serves as the foundation for the development of the use cases from which the understandings of the BMDS requirements are to be extrapolated.

21

1. Surveillance

The process of surveillance requires that sensors monitor specific geographic areas of interest for ballistic missile launch events. This implies that a commander with appropriate authority to direct the sensors that will conduct the surveillance, in response to a potential threat, has provided a specific queuing order or has determined that an area of interest warrants consistent monitoring based on high probability of an event occurring in that specific region. For instance, an increase in hostile rhetoric by a nation in possession of ballistic missiles may require that assets be committed to monitor specific regions to ensure that should a ballistic missile event occur, it would be detected with a sufficient amount of time to react accordingly. This is in contrast to certain nations that are known to possess large quantities of ballistic missiles, such as China or North Korea that will, in all likelihood, require continuous surveillance. The BMD system must manage all surveillance assets to ensure that the right assets are looking where they need to be and that there are sufficient assets for required coverage.

2. Detection

Detection is critical to any BMD system; the bottom line is that if you do not see the threat missile or know that it is coming then you cannot defend against it. Once a ballistic missile event has occurred and it has been detected by a sensor system, the BMDS must assess that what is actually being detected is in fact a ballistic missile threat and if it is, whether or not there is already a preexisting track or cueing message on that particular contact in the network. If a contact is evaluated as a threat ballistic missile and a preexisting track does not

exist, a new track must be developed and a queuing message distributed as quickly as possible so that all other participating IR and radar sensors can make their own detection and start the tracking process of the missile in flight. In doing this, the target's position can be refined through track data comparison and fusion, which can then be used to develop a weapons solution to prosecute the target. The most probable scenario is that a space-based infrared sensor will be the first to detect a ballistic missile launch and will dispatch a queuing message to other sensors that are within the field of view of the missile. As other sensors detect and track the ballistic missile they will provide their parametric data on the contact to develop a combined track through fusion and correlation.

3. Tracking

Once a missile has been detected, sensors must apply discrimination processing to ensure that what is being detected is a valid target and not an environmental anomaly, decoy, or countermeasures being conducted against the BMDS. Once the target survives this process, tracking algorithms are applied over a series of valid detections to develop a local system track where all of the target's pertinent information such as speed, altitude, range, radial velocity, geodetic positioning data, and heading can be derived utilizing information input from other systems. This information must then be stored for its own use and shared with all other units participating in the BMDS.

It is envisioned that the sensors will share the track information to develop a composite track much the same way CEC does in an effort to produce a comprehensive SIAP. By fusing the track data into a composite picture the continuity of the track is preserved, which also provides

the necessary parametric data to ensure that an intercept can be conducted by the most capable weapons platform at the earliest possible opportunity even when that weapons system is outside the field of view of the missile. Thus the sensors must develop and report tracks in an asynchronous manner; this affects the correlation and fusing of tracks. Track fusion and discrimination algorithms are currently being developed as part of Project Hercules, whose goal is to develop a composite picture and target discrimination in a high countermeasure environment.¹⁶

4. Identification

The issue of developing the capability of being able to positively identify a ballistic missile based on its performance and radar and IR signature has been on going since the advent of the SDI in the 1980's. As a process within any BMDS, because a ballistic missile is by its nature a passive object, it requires that enough information be resident within a sensor detection signal, that information can be readily extrapolated and exploited in an effort to identify the object, and that multiple sensor information can be correlated and fused to develop a positive identification of the ballistic missile.

This information would then need to be compared to a database of known missile characteristics in an effort to positively identify the type of missile. Additionally, this process would also need to incorporate the process of discrimination between a reentry vehicle and any countermeasures deployed to confuse the BMDS particularly in the midcourse phase, which will be discussed later. This process can be time consuming and computationally

¹⁶ MDA Exhibit R-2 RDT&E Budget Item Justification (PE 0603889C), Feb. 2003.

intensive and may not yield a positive identification if the signature was not extractable, the information was not evaluated properly, or a target-signature match was not possible due to inadequacies of the database.

5. Target/Engage

In order for the BMDS to be effective it must be able to place a weapon on a target. The BMDS must assess what weapons are available and are in an opportunistic position to consummate an intercept and issue launch orders in a timely enough fashion that the weapon has a possibility of making the intercept before the ballistic missile exceeds its capabilities. For instance, a SM-3 missile fired from an AEGIS cruiser at a ballistic missile that is traveling away from it must have the speed necessary to catch it before it exceeds the SM-3's maximum effective range; otherwise, the SM-3 becomes merely a wasted asset.

The BMDS must also provide target tracking that is accurate enough that a weapons system can determine its probability of destruction, which in turn will be needed by the BMDS to determine which weapons system to employ. The BMDS must also provide predictive tracking data and the lead necessary to place an interceptor kill vehicle at the proper point in space to either hit the target for a ballistic weapon or place an active homing hit-to-kill (HTK) vehicle into a position where its own organic sensor can take over and refine the intercept solution until collision.

Critical to weapons conservation is that only one interceptor should be assigned at any one time to a each ballistic missile. The BMDS must assure that only one weapon is assigned to engage a target, unless it is determined that the assigned weapon cannot consummate the

intercept or if a weapons system through system failure is operating autonomously.

6. Assess

The ability to assess the effectiveness of an intercept conducted against a threat ballistic missile is critical for conservation of limited weapons and will drive the weapons employment doctrine. Ideally, you would want to employ a "Shoot, Look, Shoot" doctrine in which an interceptor is fired at the earliest opportunity, the hit evaluated, and if unsuccessful another interceptor is launched. If the timeline is compressed or an immediate assessment cannot be made, a more liberal weapons employment approach must be used, such as "shoot, shoot, look" where multiple weapons are fired at a target until an accurate assessment can be made. The latter approach has the potential to expend many weapons early on, making it difficult to defend against additional threat ballistic missiles launched after the initial attack. However, this may be necessary if the potential for use of WMD exist and opportunities for intercept are limited. Throughout the assessment process, all of the sensors must attempt to track the target to provide feedback if the intercept was successful or not and if unsuccessful to be well prepared to continue prosecution of the threat.

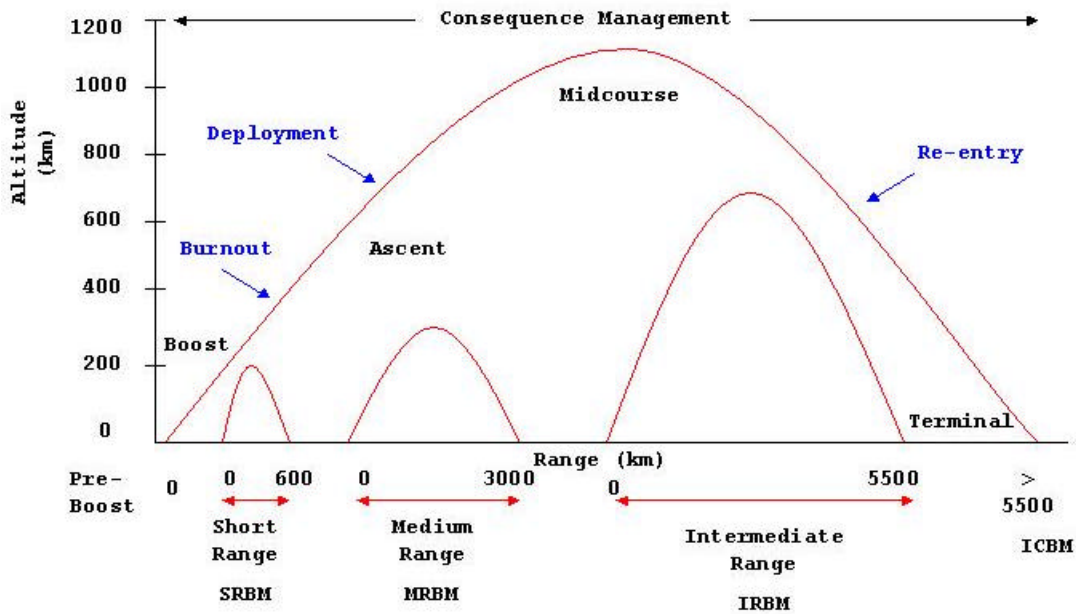


Figure 2. An Integrated, Layered Defense Against Missiles of All Ranges

C. PHASES OF FLIGHT

A ballistic missile has three distinct phase of flight: the boost, midcourse and terminal phases (Figure 2). Each one of these phases has certain advantages for conducting an intercept while the missile is in each particular region and some significant disadvantages that will need to be minimized if possible in order for the BMDS to consummate an intercept. We will need to look at each phase to determine what ramification each phase has on timing considerations in developing a system-of-systems.

1. Boost Phase

This region of flight is the most desirable for conducting an intercept of a threat ballistic missile. The missile is traveling at its slowest rate of speed during the boost phase, the large IR signature caused by the

launch plume is usually easier to detect provided that there is no obscurant such as weather, and most importantly, if a ballistic missile is intercepted and destroyed in this phase, in all likelihood the warhead and debris will fall well short of the intended target and hopefully on the territory of the nation that launched the missile.

However, this is also the most difficult phase in which to intercept a ballistic missile and will require perhaps the greatest level of automated decision making due to the short duration of this phase. It is estimated that the engagement time for a weapons system during the boost phase varies from one minute for a short-range ballistic missile up to four minutes for an ICBM, which is our primary focus of concern in this thesis. While the missile starts at zero speed and is most vulnerable due to lack of speed for maneuverability, and is more easily detectable due to the size of the entire missile and the heat signature the booster produces, it is continuously accelerating until booster burnout and detachment of the reentry vehicle, making it harder to hit as time passes (this phase is also referred to as the post-boost phase in some literature)¹⁷. This also makes the kill chain timeline short due to the need of detecting and identifying the threat missile properly, therefore necessitating a relatively high reliance on automation of battle management. Adding into the equation natural phenomenon such as weather and the ability to launch a weapon well within one territorial boundary beyond the detection of local active sensors and reach of weapons systems, the

¹⁷ BMDO, *Harnessing the Power of Technology, The Road to Ballistic Missile Defense From 1987-2007*, Sep. 2000, p 6.

difficulty to conduct an intercept of a ballistic missile in the boost phase of flight becomes even more difficult.

2. Midcourse

The midcourse phase of flight is defined as that portion where the missile has departed endoatmospheric and travels in the exoatmospheric region. It is during this phase of the missile's flight which provides the greatest opportunity to engage with an interceptor based on the length of time that the missile is in this phase, generally an ICBM remains in this phase of flight up to twenty minutes prior to entering the reentry or terminal phase. In addition to the extended period of time to conduct an intercept, the missile is also in a coast mode with no additional source of power and maintains a relatively predictable path. Therefore the ability to launch multiple interceptors from geographically dispersed locations, and the ability to assess the outcome of those launches is increased. The increased time available also allows for the utilization of the look, shoot, look doctrine enhancing the probability of destruction in this phase.

The down side to this phase of flight is that the incoming missile can deploy countermeasures that the BMDS must be able to discriminate against to avoid track saturation and detect the actual reentry warhead. This has been identified by MDA as perhaps the most difficult challenge to overcome in developing and implementing an effective BMDS.

3. Terminal

The terminal phase begins at the point at which the reentry vehicle enters the endoatmospheric region. This phase is short: approximately thirty seconds, due to the velocity of the reentry vehicle and the effects of gravity. The advantages of this phase are that the reentry vehicle

is still on a predictable path and discrimination between countermeasures and actual warhead is much more feasible than in the other phases of flight. This is due to the fact that if any countermeasures were deployed, they would more than likely weigh less than the warhead and would slow down or possibly burn up on reentry, thus unmasking the actual warhead increasing the probability of destruction of the actual target.

The major disadvantages of trying to engage during the terminal phase are that the time line is short: reaction time is limited as is the number of interceptors that can be deployed against the reentry vehicle. Add into this equation the closure speed of any interceptor and the velocity of the missile, the decision of when to launch is relatively more time critical than the other phases. This time line can be even shorter if the missile is of the short or medium range variety that does not enter the exoatmospheric regime. Also, as happened in Desert Storm, even if a ballistic missile is intercepted, the debris will likely fall on friendly territory.

D. BMDS COMPONENTS

A major defensive system can be broken down in three primary elements that perform the functions of the kill chain throughout all the phases of the ballistic missile's flight; these are the sensors, weapons, and C2. Each of the components within the BMDS has unique requirements both in the way it conducts systems operation and processing of data, the frequency of the communication mode and data throughput, the overhead of encryption and decryption, and the distance that it must transmit and receive the data.

1. Sensors

The BMDS will need to consist of air-, space-, ground-, and sea-based sensors to provide as complete a level of coverage as possible to detect, track, and report ballistic missiles through all phases of flight (Figure 3). These will consist of passive optical, infrared, and ultraviolet sensors, primarily in space, LADAR (laser detection and ranging) on aircraft, and active radars on board aircraft, ships, and ground-based locations.¹⁸ These sensors will be able to operate autonomously or as envisioned as a system-of-systems layered and networked providing the most current and accurate track data available to all participating units within the BMDS architecture.

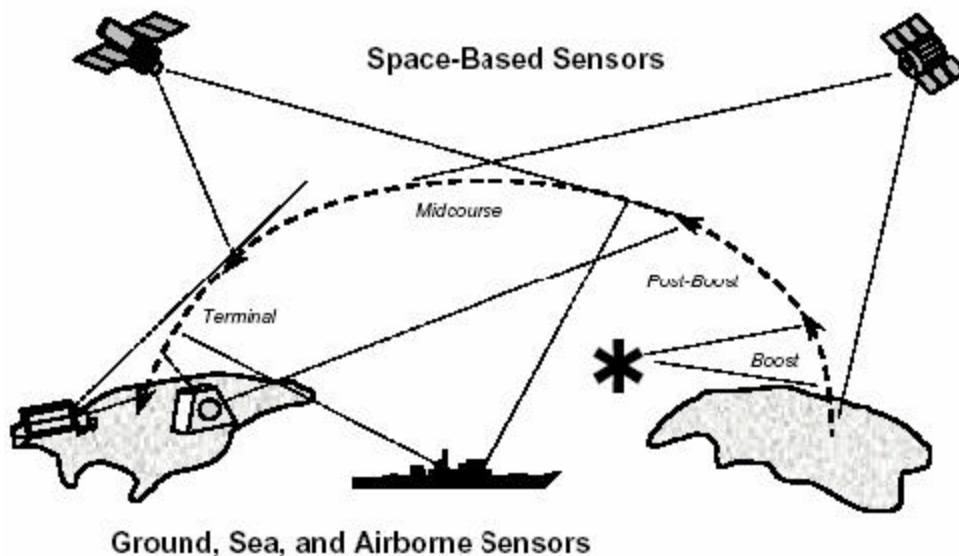


Figure 3. BMDS Sensor Diagram.¹⁹

Currently, the primary space-based passive IR BMDS sensors are the older DSP satellites; they will start to be replaced by the Space Based Infrared System-High (SBIRS-High). SBIRS-High will consist of six satellites, four in

¹⁸ Ibid. p 8.

¹⁹ Ibid. p 8.

geosynchronous orbit (GEO) and two in highly elliptical orbits (HEO), also known as Molniya orbit, and their sensors will cover short-wave infrared, expanded mid-wave infrared and see-to-the-ground bands, allowing it to perform a broader set of missions as compared to DSP.

SBIRS-Low, which is the critical component of SBIRS, will consist of twenty-four Low Earth Orbit (LEO) satellites providing a unique precision boost, midcourse, and reentry tracking capability, and providing decoy discrimination data that is critical for effective BMD. The Low satellite's sensors will operate across long and short wave infrared, as well as the visible light spectrum. The long wave infrared (LWIR) spectrum is unique to the Low system in that it will allow cold-body tracking of a missile in the mid-course phase of flight.²⁰

The SBIRS concept provides a synergistic approach to the detection of ballistic missiles by distributing sensor tasking which will prevent overloading by a single satellite and allow multiple satellites to track targets improving target data and providing continuous tracking of a BM from launch through reentry. Both the SBIRS-High and SBIRS-Low will have the ability to detect launches and will be able to handoff a target to another satellite as the threat missile leaves its field of view and can cue ground-based radars while the threat missile is still below the radars' horizon.

In considering timing issues with space-based IR sensors, the detection is passive and in one direction, that is energy is transmitted by the missile and detected by the IR sensor. Therefore, the time it is merely a function of the range of the ballistic missile from the satellite. This is summated with the total processing time

²⁰ <http://www.fas.org/spp/starwars/program/dote99/99sbirs.htm>.

and the time to transmit the track to other sensors within the network.

As previously mentioned, the active detection systems within the sensor category will consist of ground-, air-, and sea-based radars. Active sensors will generally require more time overall to develop a track due to the need for transmitting a signal out and receiving a reply, that is, the range of the sensor to the target squared, in addition to the processing and transmitting of a track. However, this time is shortened when the target is in close proximity to the sensor. In order to conduct surveillance and target tracking of ballistic missiles, radar sensors require large amounts of power; that is why space-based radar sensors are not currently envisioned for use. However, for the purposes of simulating a system-of-systems, inclusion of such capability needs to occur to validate the concept of this approach.

The current plan for sea-based radar systems is to provide upgrades to the AEGIS systems on board ships to bring them up to BMDS SPY-1D capabilities. In conjunction with the SM-3 missiles, AEGIS systems will be able to provide midcourse phase surveillance, tracking, and intercept capabilities as part of the BMDS. However, the sea-based BMDS, by its very nature, provides additional time considerations that must be addressed. This is due to the fact that for midcourse interception to occur, particularly for all but possibly short-range ballistic missiles, the AEGIS system will more than likely be out of a direct line-of-sight communications path with other BMDS participating units. This implies that any AEGIS ship must transmit and receive via RF satellite communications, in effect doubling the communications distance, in addition to

the time required for active detection and track processing.

Ground-based radar (GBR) systems will consist of older but Upgraded Early Warning Radars (UEWR) and new X-band Radar (XBR) systems both of which will track and provide initial planning for an intercept primarily in the midcourse and terminal phases of flight. The UEWR are currently in operation and will be used primarily for surveillance, detection and tracking of ballistic missiles.

The XBR systems, in addition to conduct standard active radar functions, will also possess the capability to provide primary fire-control information for ground-based interceptors (GBI), provide discrimination among warheads and countermeasures or decoys, providing this information to the GBI, and kill assessment of the targeted threat ballistic missile. Plans currently call for an XBR to be outfitted on board a sea-based platform, similar in appearance to a floating oil rig, to be stationed in Adak, Alaska with follow on systems to be field at other locations.

The GBR systems currently in use and future systems will need to be place in remote locations such as Alaska, coastal areas, and in the northern Midwest in order to detect the shortest and most likely avenue of approach of ballistic missiles targeted against the U.S. These systems will have considerably longer radar detection ranges at maximum distance. The communications path, however, while not shorter in distance for transmission, will have high bandwidth data-transmission capability.

There currently are no air-based radar, infrared, or LADAR capabilities that can track a ballistic missile consistently throughout all phases of flight, and with limited capability in the exoatmospheric regime. There are

developmental systems, such as radar upgrades for both AWACS and E-2C aircraft, and IR and LADAR packages for surveillance aircraft as well as unmanned aerial vehicles (UAV) that provide promise for detection and tracking capabilities in the boost and terminal phases of flight.

2. Weapons

The key to any successful defensive system is the ability to negate the threat either through deception or destruction. BMDS is no exception and there are numerous weapons programs currently under development with each being designed to intercept a ballistic missile in one or more phases of flight from geographically disparate locations (Figure 4).

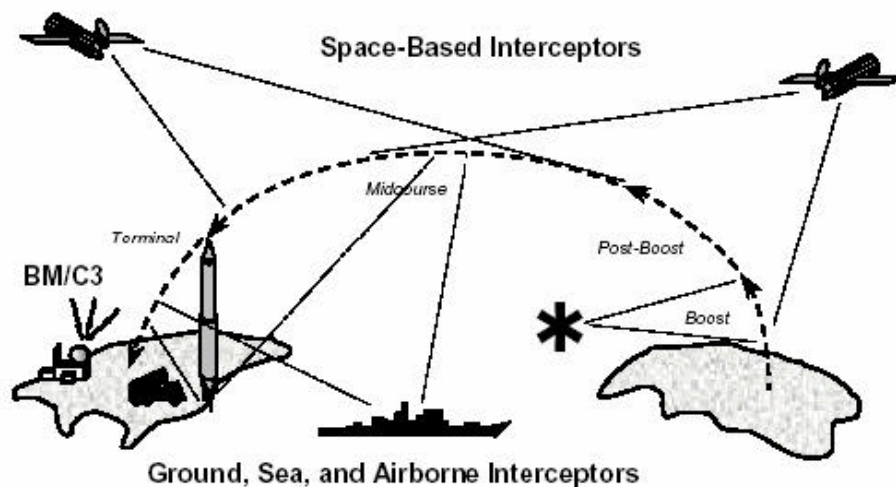


Figure 4. BMDS Interceptor Diagram.²¹

Critical to the effectiveness of each of these weapons systems is that they must be in a position to consummate an intercept and deliver enough energy to destroy the threat. The more accurate the weapons are the less energy that will be required to destroy it. The types of weapons that are currently being pursued are HTK interceptor vehicles. With

²¹ BMDO, *Harnessing the Power of Technology, The Road to Ballistic Missile Defense From 1987-2007*, Sep. 2000, p 9.

technological advances allowing size and weight reductions and removing the explosive warhead (either conventional or nuclear) of the interceptors has been translated into a higher degree of accuracy, greater thrust, agility, and lower cost.²²

There are existing weapons systems that are currently being modified to support the BMDS concept. As mentioned earlier, the AEGIS system with the SM-3 HTK missile possessing a LWIR seeker head will provide sea-based midcourse-interception capability, which can also be utilized in the boost and terminal phase if the ship is so positioned. The Patriot Advanced Capability-3 (PAC-3) system will support greater terminal-phase interception capability providing a layered defense in conjunction with the Theater High Altitude Area Defense (THAAD) System, which in turn will address the short- and medium-range threat at high altitudes.

Under development is the Ground-Based Interceptor (GBI), which is a HTK vehicle designed to conduct intercepts of ballistic missiles from mid-course to terminal phases. Once the missile is launched, it receives guidance from the BMC3 until it can actively acquire the threat missile with its own sensor. Once the GBI starts active tracking of the threat, it will prosecute the target on its own with the ability to conduct discrimination of countermeasures and decoys.

In an effort to conduct intercepts in the boost phase the Airborne Laser (ABL) is being developed. A chemical laser placed on board a 747, its mission is to conduct boost-phase intercepts by directing a high-energy laser onto a vulnerable portion of a ballistic missile causing

²² Ibid, p 9.

intense heat and explosion. Because the ABL is designed to destroy ballistic missiles in the boost phase, the aircraft must be close enough to the launcher to employ the laser within its effective range and must have the most up-to-date and accurate information available in order to target the laser beam on to a specific portion of the missile where the damage will be the greatest. This same concept is also being pursued for a similar space-based system, although current technology has not advanced to the point that can make this project realizable in the near future.

3. Command, Control, Battle Management, and Communications (C2BMC)

Integral for any of the aforementioned weapons systems to be effective is the ability to accurately predict a ballistic missile's path based on historical track data and quickly determine the most capable and opportunistic weapon to employ. The C2BMC must be able to perform these functions quickly, as well as integrate and coordinate with other C2BMC nodes within the BMDS and provide direction, orders, and controls to subordinate sensors and weapons, regardless of the number of units under the C2BMC control. As a threat ballistic missile transits from one sensors field of view to another, the C2BMC must ensure a timely and positive handover and cue other local, bordering C2BMC nodes and higher command elements of the presence of threat ballistic missiles. In essence the C2BMC must be the arbitrator and resources manager for its designated area of responsibility (AOR), performing the following tasks: creating a defensive plan, optimizing overall defense, utilizing all resources, controlling systems according to unified concepts, improving overall defense structure efficiency, and minimizing overkill phenomenon.²³

²³ Haim Baruch, *Battle Management*, AIAA, 2000, p 207.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. BMDS REQUIREMENTS SPECIFICATION

A. INTRODUCTION

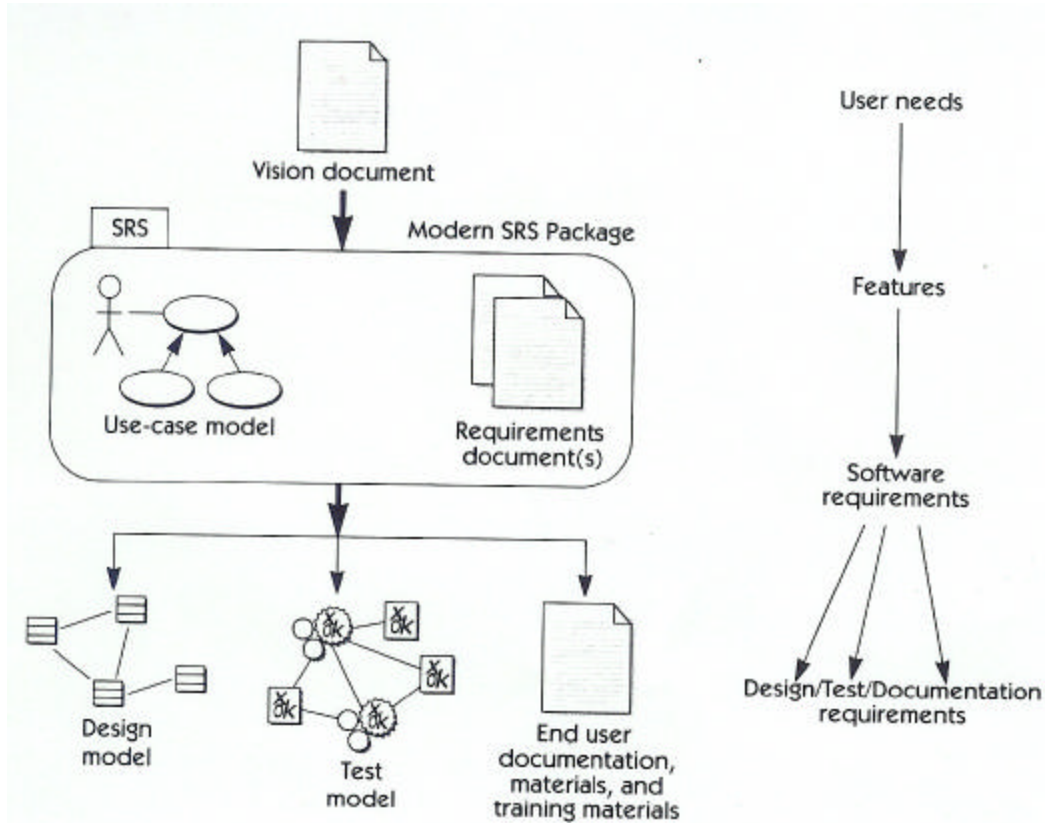


Figure 5. Process of BMDS development²⁴

From the onset, we envision that the BMDS, as a system-of-systems, will be designed with high cohesion with low coupling because of both its requirements for distributed and real time prosecution of ballistic missiles.²⁵ Additionally, as previously mentioned in Chapter II, the Presidents vision is to implement systems

²⁴ Dean Leffingwell, Don Widrig, *Managing Software Requirements*, Addison-Wesley, 2000, p 263.

²⁵ Caffall, D. S. and Michael, J. B. "A new paradigm for requirements specification and analysis of system-of-systems". Wirsing, M., Balsamo, S., and Knapp, A., eds., *Lecture Notes in Computer Science: Proc Monterey Workshop 2002: Radical Innovations of Software and Systems Engineering in the Future*, Berlin: Springer-Verlag, 2003.

as they are developed over time which necessitates this methodology to prevent the pitfalls of stove piping.

We have approached the problem of developing the requirements of such a system-of-systems by utilizing standardized software engineering processes, as depicted in Figure 5, in an effort to determine both the high level requirements of the BMDS and to flush out the potential timing constraints on the system.

B. VISION AND SOFTWARE REQUIREMENT SPECIFICATION (SRS) DOCUMENTS

The first stage in the continuing development of the requirements specification for the notional BMDS was to create a vision document (Appendix B) based on the previous work completed.²⁶ The purpose of this document is to establish a starting point for the project that capture the needs of the user, in this case MDA, the initial features and high-level capabilities of the system, some of the high-level requirements, and definition of the problem and solution at a high-level of abstraction. Utilizing a vision document template we were able to use the kill chain process to determine foundation for the use case diagrams and the high-level functionality of the BMDS that needed to be realized. The vision document will need to be modified as time passes to reflect a more refined vision of what the system should be and continuously referred to throughout the process of system development. This will ensure that the requirements of the system that have been identified in the use cases can be traced back to the vision document, providing an indication that the necessary features of the system are being addressed.

²⁶ Dale Scott Caffall, "Conceptual Framework Approach for System-of-Systems Software Developments" (M.S. Thesis, Naval Postgraduate School, Mar. 2003).

As the requirements of the BMDS are identified and documented, they will be refined into the Software Requirements Specification (SRS) package (Appendix C). The fundamental differences between the SRS and the vision document is that the vision document is a broad-based description of the users' needs, goals, objectives, and system features, whereas the SRS describes how these features are to be implemented and the external behaviors of the system in order to develop a solution to the software development problem. The primary purpose of the SRS is to serve as a reference standard for the development team encompassing the functional and nonfunctional requirements and design constraints of the system controlling the project evolution through the input of the design, implementation, and testing groups. As the vision document evolves, the SRS must reflect those changes and serve as a traceability reference point for verification and validation testing to ensure that the developed system is meeting the established requirements.

C. DESCRIPTION OF BMDS ARCHITECTURE²⁷

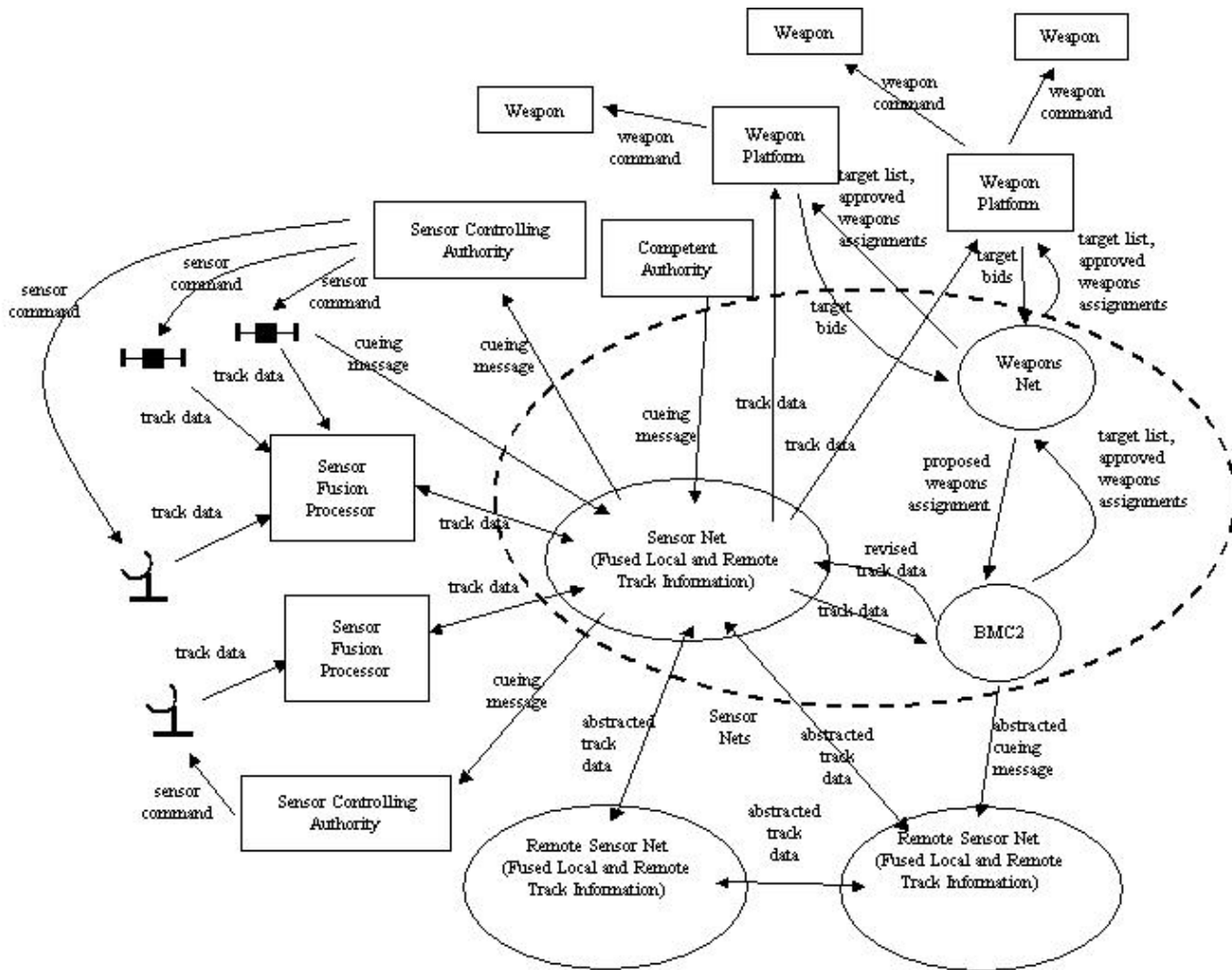


Figure 6. Distributed C2BMC Architecture.

A distributed system is defined as:

...one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages...Computers that are connected by a network may be spatially separated by any distance. They may be on separate continents, in the same building or in the same room...distributed

²⁷ James Bret Michael, Phillip Pace, Man-Tak Shing, Murali Tummala and others, eds., *Test and Evaluation of the Ballistic Missile Defense System FY 03 Progress Report*, Naval Postgraduate School, Sept. 2003.

systems has the following significant consequences: concurrency; no global clock; independent failures. The motivation for constructing and using distributed systems stems from a desire to share resources.²⁸

The complexity, size, and the need for concurrency of a global BMDS necessitates that it be developed as a distributed system. In order to achieve the end goal of developing a distributed system-of-systems we envision the BMDS as depicted in the high-level distributed architecture as shown in Figure 6.

The overarching BMDS will consist of a loosely coupled set of regional C2BMC systems; geographically separated networks interconnected much like the Internet. The intent is to allow all participants to pull the information from specific areas of responsibility (AOR) as desired, but also to ensure that time-critical information can be pushed to those geographically collocated units that need it to effect destruction of a threat ballistic missile or to hand-off the information to non-geographically collocated units as a missile transits from one region to another. Note that the various sensors and weapons may be connected to more than one regional C2BMC system via proxy. The advantage is that geographic location is a "don't care" in that context.

The real-time nature of the battle requires that all sensor information be local to fight the battle. As the missile continues in its flight, the real-time battle management, together with some of the sensors and weapons, will handover to another regional C2BMC system. The use of

²⁸ George Coulouris, Jean Dollimore, and Tim Kindberg, *Distributed Systems Concepts and Designs*, Addison-Wesley, New York, 2001, p 2.

the Broker pattern will ease the handover of the assets from one region to another.²⁹

By distributing the C2BMC in this manner, information regarding any ballistic missile threat is available and accessible to all participants as desired, but will not overburden the network by having all the information presented to all units all the time; this will provide increased availability of data; more localized control, and improved response times of the units to counter the threat. Thus, units subscribe to the network with their addresses being available in routing tables with knowledge of the geographic location of the unit so that only data and information relevant to a particular unit (or region) is forwarded to that unit (or region). For example, fire-control data from another theater or region may not be useful and hence will stay local, while threat information from other theaters or regions may provide valuable situational awareness and therefore it can be made available to other regions. Each regional BMC2 system consists of three major sub-systems: a Sensor Net, a Weapons Net and a BMC2.

Sensor Net refers to a distributed system that provides the sharing of track data among Sensor Fusion Processors, Weapons Net, Weapon Platforms and the BMC2. It supports a distributed track data-bidding process through which the Sensor Fusion Processors collaboratively perform track correlation along with fusion to improve the quality of the integrated air picture. It also allows the broadcasting of cueing messages among Competent Authorities and the Sensor Controlling Authority.

²⁹ F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*. Wiley & Sons, New York, 1996.

Weapons Net refers to a distributed system for target bidding. It manages a list of targets waiting to be engaged by the Ballistic Missile Defense System, and coordinates cooperative weapons assignments (i.e., the pairing of appropriate weapons with targets) based on the bids (i.e., figure of merits that are based on many factors such as the defended area, predicted impact point, threat type, health and status of weapons, current engagements) submitted by individual weapons platforms, and policies, rules of engagement and manual overrides from the battle manager.

BMC2 refers to the automation for supporting the BMC2 functions. It provides the interface for battle managers to create, modify, or delete the prioritized target list, set the initial weapons authorizations and other rules of engagement, and to monitor the engagement to its conclusion given that it may have to reassign the track to another weapon.

The regional BMC2 will be supported by three integrated sub-networks: a Sensor Net, a Weapons Net, and a Command and Control (C2) Net emulating a geographical intranet. The primary justification for the division along functional lines is that the data, in its entirety, flowing across each network may not be relevant to the others. For instance, the specific radar parametric data derived from the Sensors is critical for use in the Weapons Net but is not necessary for C2; only the particular missile-track information (e.g., a Link-16 track) is pertinent. Conversely, intelligence information, such as electronic intelligence (ELINT) or human intelligence (HUMINT) regarding the possible numbers, location or movement of missiles that is critical for C2 planning is generally not

critical for the actual employment of a weapon system or to conduct sensor tracking. Therefore, the data that is critical for each network will be determined and made available, but information that is not critical for the functional area will not be provided, thus preventing excessive overload on that particular network that doesn't require the data.

At the Sensor and Weapons Net, the message format will need to be binary and in a standardized format to reduce overhead and time latency, and ensure time-critical data is made available to the participating units that need it. The C2 network, by necessity, will consist of more than just track data to include, United States Message Text Format (USMTF) messages, intelligence data, etc. Current C2 systems incorporate middleware such as XML or CORBA in order to integrate legacy sensor and weapons systems to keep the implementation independent of the platform. It is our desire to move away from this scheme.

The BMC2 System will need to consist of various communication mediums in order to connect the various participants operating with heterogeneous communications suites. Currently, MDA is considering fiber-optic cable for the terrestrial elements of the network and will allow large throughput of data. However for the air-, sea-, and space-based elements the only possible means for data transmission is by RF energy. For space-based systems, UHF, EHF, or SHF can be utilized and the obvious choice would be the higher frequencies for greater data throughput. Ground-, and sea-based units can also utilize these frequency ranges. However, due to the higher frequencies requiring large antenna sizes, only the larger combatants ships will be able to participate at the EHF/SHF

level. For air-based units, the only viable choice currently for data transmission is in the UHF frequency band. This is driven by the need for smaller antenna sizes and only UHF has a high enough data throughput to be effective. The bottom line is whatever platform the servers reside on, they will need to be capable of transmitting and receiving data from all sources.

As mentioned previously, each of the nets are divided along functional lines and will consist of the data necessary to conduct their primary mission. The C2 Net will be interfaced with the Sensor and Weapons Net to provide C2 functionality for the direction and employment of each of these systems. The sensors will be cued by command inputs from the Sensor Net via the C2 Net and track data will be received for distribution to higher and adjacent command elements interfaced within the C2 Net and BMC2 system. Weapons systems assignment shall be directed for employment based on the tracking-data inputs from the Sensor Net, weapons availability from the Weapons Net, and the previously mentioned aspects of the weapons tasking logic. The C2 Net will be interfaced with higher and adjacent commands in the BMC2 system for coordination and information exchange, such as the hand-off of tracks.

The Weapons Net will encompass all participating weapons systems. A bidding process shall occur for the employment of weapons on specifically designated targets provided by the Sensor Net. The weapons bidding process will be the basis of weapons assignment, thus precluding expending multiple weapons from different weapons platforms on one target. As envisioned, each weapons system will evaluate the tracks provided by the Sensor Net, determine a numerical value based on the trajectory of the missile and

its evaluation of the probability of kill, and then place a bid. After a predefined amount of time, the bidding will be locked and a weapons assignment (i.e., the pairing of weapons and targets) will be made using a three-phase commit protocol that is able to tolerate both site and communications failure, while minimizing the frequency of blocking below that of two-phase commit.³⁰ Each weapon system will continue to evaluate the target in the eventuality that the weapon misses or does not completely destroy the target. If the target is destroyed the process is complete, else the bidding process starts anew. The battle manager continuously oversees the whole process, following each track through the entire engagement process.

The Sensor Net consists of netting all of the available sensors for the detection of a ballistic missile in a regional BMC2 system. Each sensor, as it develops a track on a ballistic missile, will transmit the track to a Sensor Fusion Processor. The tracks that are developed and transmitted by the sensors will carry a timestamp along with the target's parametric data so that the Sensor Fusion Processor will be able to utilize the most current information with which to update the track. At the Sensor Fusion Processor, the data from its local sensor sources will go through an initial discrimination scheme. A track table will need to be maintained on each contact. As each track report arrives, it will need to be correlated based both on an evaluation of the contact current positional status in relation to tracks from other sensors, and a comparison of its current position in relation to a calculated predictive parametric behavior. This will ensure that the contact is

³⁰ Philip Bernstein, Vassos Hadzilacos, and Nathan Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987, p 240.

valid and can be updated by the most current source, and validate that it is the actual missile and not a decoy, debris, or perhaps another missile in close proximity. Once the missile contact is validated, the Sensor Fusion Processor will develop a single track containing the pertinent target data and a unique identifier. The fused track will be pushed onto the Sensor Net for utilization by all participating units. The pertinent parametric data will also be pushed to the Weapons Net for weapons system utilization and weapons bidding. The track data will be also pushed to the C2 Net for situational awareness and command and control decision-making.

D. USE CASES

A use case is defined as:

A description of a set of sequences of actions, including variants that a system performs that yields an observable result of value to an actor.. when used in the context of system development the Use Cases establish the desired behavior of the system for verifying and validating the system architecture. ³¹

The use cases, in other words, identify who, what and how of system behavior through the interactions between a user and that system.³² Five use cases for the BMDS, each corresponding to a different phase of the kill chain, have been proposed.³³ In our thesis, we refine those use cases to identify system requirements, behaviors, and timing

³¹ Grady Booch, Jim Rumbaugh, Ivar Jacobson, *The UML Reference Manual*, Addison Wesley 1999, p 488.

³² Dean Leffingwell, Don Widrig, *Managing Software Requirements*, Addison-Wesley, 2000, p 235.

³³ Dale Scott Caffall, "Conceptual Framework Approach for System-of-Systems Software Developments" (M.S. Thesis, Naval Postgraduate School, Mar. 2003).

constraints (Figure 7). These use cases will become part of the SRS, and will need to be periodically refined. This process will be complete when there are sufficient enough use cases that can describe all possible ways in which the system can function. When this is achieved these Use Cases will then serve as the foundation for further design, implementation, and testing of the system.

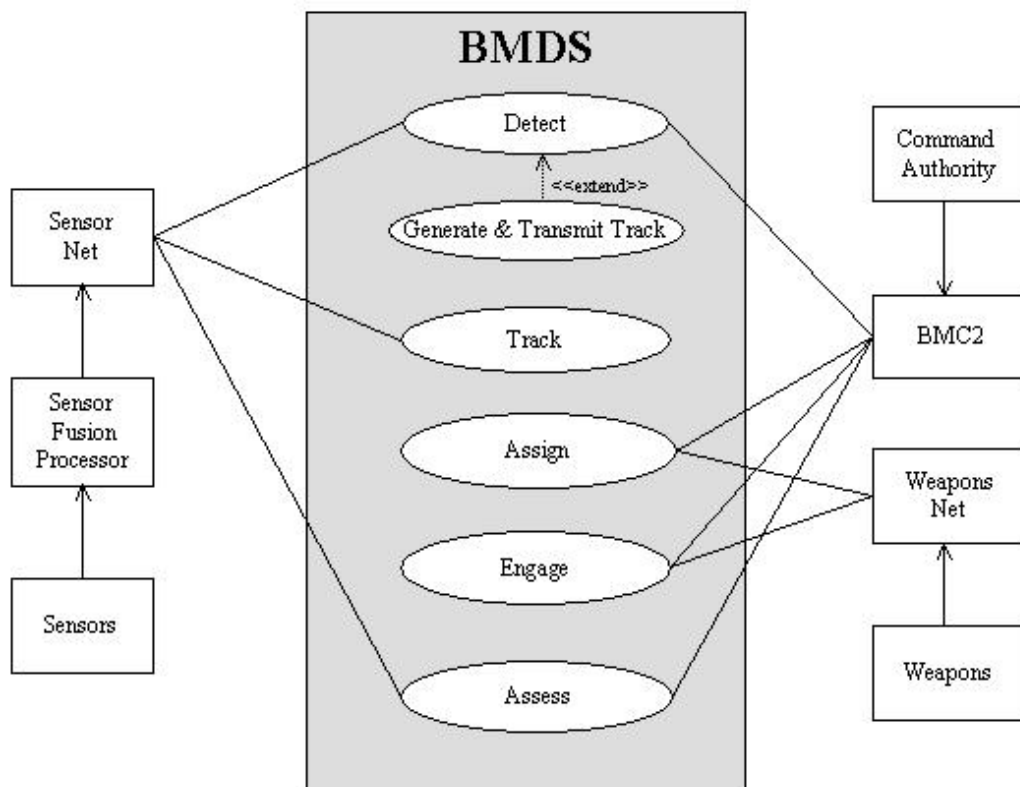


Figure 7. High-Level BMDS Use Case

1. Use Case 1: Detect Potential Threat Ballistic Missile.

Context Diagram:

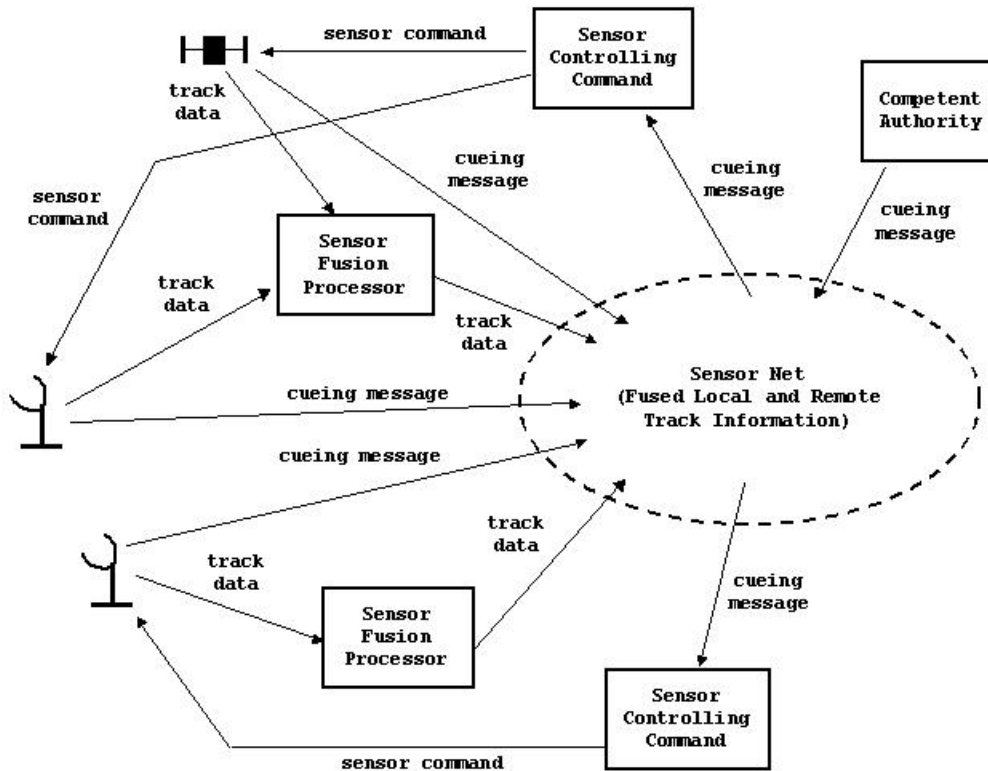


Figure 8. Use Case 1 Diagram

Context of Use: The goal of this use case is to detect possible threat ballistic missile and push the track data onto the Sensor Net.

Level: User goal.

Primary Actors: Threat ballistic missile, Sensor Net, Sensor Fusion Processor, Sensors, Sensor Controlling Authority, Competent Authority.

Stakeholders and Interests: Regional Commanders, Higher Commanders.³⁴

³⁴ Higher Commanders is defined as all those commanders senior to and in the direct chain of command of the regional commander.

Preconditions: Sensor is in search mode.

Success Guarantee: Sensor Fusion Processor develops a single-track file for the potential threat ballistic missile.

Trigger: Adversary launches threat ballistic missile.

Main Success Scenario:

Competent authority determines that a potential ballistic missile threat exists in a predetermined geographic region, and issues cueing command message to Sensor Controlling Authority via the Sensor Net to position sensors in such a way that will allow a potential threat ballistic missile event to be detected within the field of view of the sensors.

Sensor Controlling Authority receives cue from Sensor Net and directs sensors towards potential threat.

Individual sensor initiates Use Case 1.1 to develop a local track for the potential threat ballistic missile and transmit track files to the Sensor Fusion Processor.

Sensor Fusion Processor receives one or more tracks and filters data from its associated sensors, and develops single-track file for the potential threat.

Extensions:

1a: If potential threat ballistic missile is not determined to exist in the area of interest then no cueing message will be issued.

2a: If Sensor Controlling Authority receives no cue, the sensor will continue to conduct surveillance in its current region.

2b: If Sensor Controlling Authority receives cueing message but is unable to comply with the cueing message from the Competent Authority, a Non-Compliance Message shall be forwarded to the Competent Authority and the sensor will continue to conduct surveillance in its current region.

3a: If none of the sensors generates a track file for the potential threat ballistic missile, then the process of detecting a threat ballistic missile fails.

4a: If the sensor fusion processor received no track file from sensors, then the process of detecting a threat ballistic missile fails.

Technical and Data Variations List: None

2. Use Case 1.1: Generate and Transmit a Local Track

Context of Use: The goal of this use case is to have a sensor generate a local track based on valid detection parameters of the sensor.

Level: Sub-use-case of Use Case 1.

Primary Actors: Threat ballistic missile, Sensors, Sensor Net, Sensor Fusion Processor

Stakeholders and Interests: Regional Commanders, Higher Commanders

Preconditions: A potential threat ballistic missile event to be detected within the field of view of the sensor.

Success Guaranteed: Sensor develops and transmits an active potential threat ballistic missile track to Sensor Fusion Processor

Trigger: A potential threat ballistic missile event has occurred within the field of view of the sensor.

Main Success Scenario:

Sensor observes a potential threat ballistic missile event that meets or exceeds the sensor's detection threshold within its field of view and develops a hit.³⁵

Sensor generates cueing message, providing precise location as to where the event is taking place and transmits it to Sensor Net.

³⁵ "Develop a hit" is defined as a sensor signal that survives the sensors environmental and false detection processing.

Sensor starts tracking to develop and refine the hits into a singular, coherent track when the number of hits exceeds the track threshold.

Sensor transmits the track data to the appropriate Sensor Fusion Processor.

Extensions:

1a: If data is not sufficient to pass screening, then the detection process fails. Neither track nor cueing message are generated. The sensor will continue to monitor the environment.

2a: If precise location is not attainable, the sensor will provide sufficient data to cue remote sensors to a general locale for surveillance.

3a: If the sensor has detected an event but the number of detections does not exceed the track threshold, the process fails. No track will be generated. The sensor will continue to monitor the environment.

Technical and Data Variation List:

Track information shall include track-identification value, time stamp, track quality, geo-reference, missile identification, bearing, altitude, direction of travel, speed, and parametric sensor-data information.

3. Use Case 2: Cooperatively Track and Classify Threat Ballistic Missiles

Context Diagram:

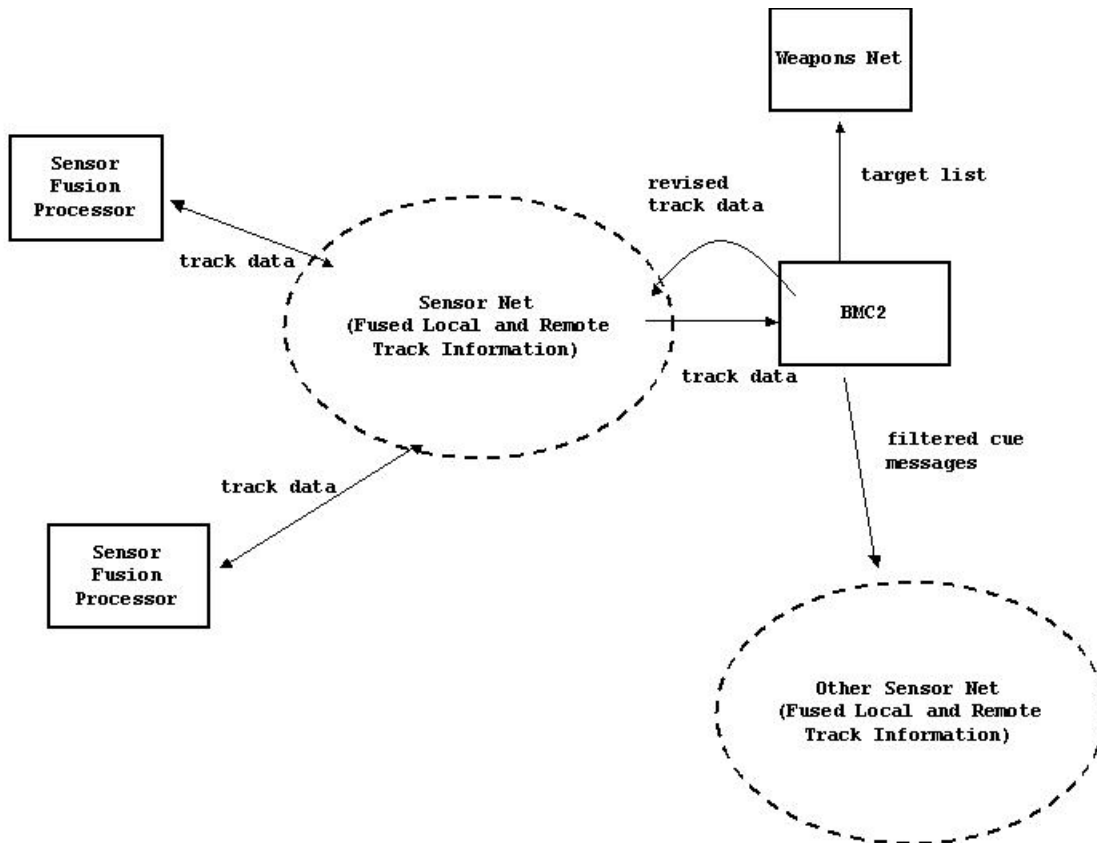


Figure 9. Use Case 2 Diagram

Context of Use: The goal of this use case is to identify and type-classify the threat ballistic missile, develop fire-quality tracks for engagement solutions, and forward target-track list to Weapons Net.

Primary Actors: Sensor Net, Sensor Fusion Processors, Weapons Net, BMC2

Stakeholders and Interests: Regional Commanders, Higher Commanders

Preconditions: Sensor Fusion Processors, Sensor Net, and Weapons Net are all operational.

Success Guarantee: BMC2 forwards target track list to the Weapons Net.

Trigger: Sensor Fusion Processors are tracking potential threat ballistic missile(s).

Main Success Scenario:

Individual Sensor Fusion Processor uses intelligence profiles of threat ballistic missile(s) to type-classify tracks.

Individual Sensor Fusion Processor provides type-classified track data to Sensor Net.

Individual Sensor Fusion Processor compares the track data in the Sensor Net against its own developed and improved track data by fusing data obtained from other Sensor Fusion Processors with its own and adding cross-references to those tracks in the Sensor Net. The Sensor Fusion Processor then forwards the improved track data to the Sensor Net.

Situation Awareness Filters within the BMC2 monitor tracks in Sensor Net, and develop and forward cueing messages to neighboring Sensor Nets.

The Target List Coordinator within the BMC2 develops one master target list and forwards it to Weapons Net.

Extensions:

1a: If all Sensor Fusion Processors determine that the track is not a threat, the process fails.

1b: If a Sensor Fusion Processor fails to type-classify a track, it will label it as "unknown." The BMC2, which monitors those tracks resident in Sensor Net, will attempt to re-classify the "unknown" track as "hostile,"

"friendly," "neutral," "assumed friend," or "assumed hostile."

3a: If a Sensor Fusion Processor fails to produce improved track data then that data which is obtained from other Sensor Fusion Processors, it will stop sending its own track data (which will not result in better quality tracks) to the Sensor Net until such a time that it can produce better quality tracks than what exists on Sensor Net.

3b: If the Sensor Fusion Processors fail to merge tracks, then multiple tracks for the same target will appear in the Sensor Net.

4a: If Situation Awareness Filters fail to forward cueing messages to neighboring Sensor Nets, sensors in neighboring regions will continue to conduct surveillance in their current regions.

5a: If the BMC2 fails to develop a target list and forward information to Weapons Net, process fails.

Technical and Data Variations List:

Sensor Fusion Processors and BMC2s will have electronic access to intelligence profiles of threat ballistic missile.

Fire-quality track information shall include position, velocity, covariance, sigma, missile type, impact point prediction (IPP), launch point estimate (LPE), and re-entry vehicle (RV) type.

4. Use Case 3: Cooperative Weapons Assignment

Context Diagram:

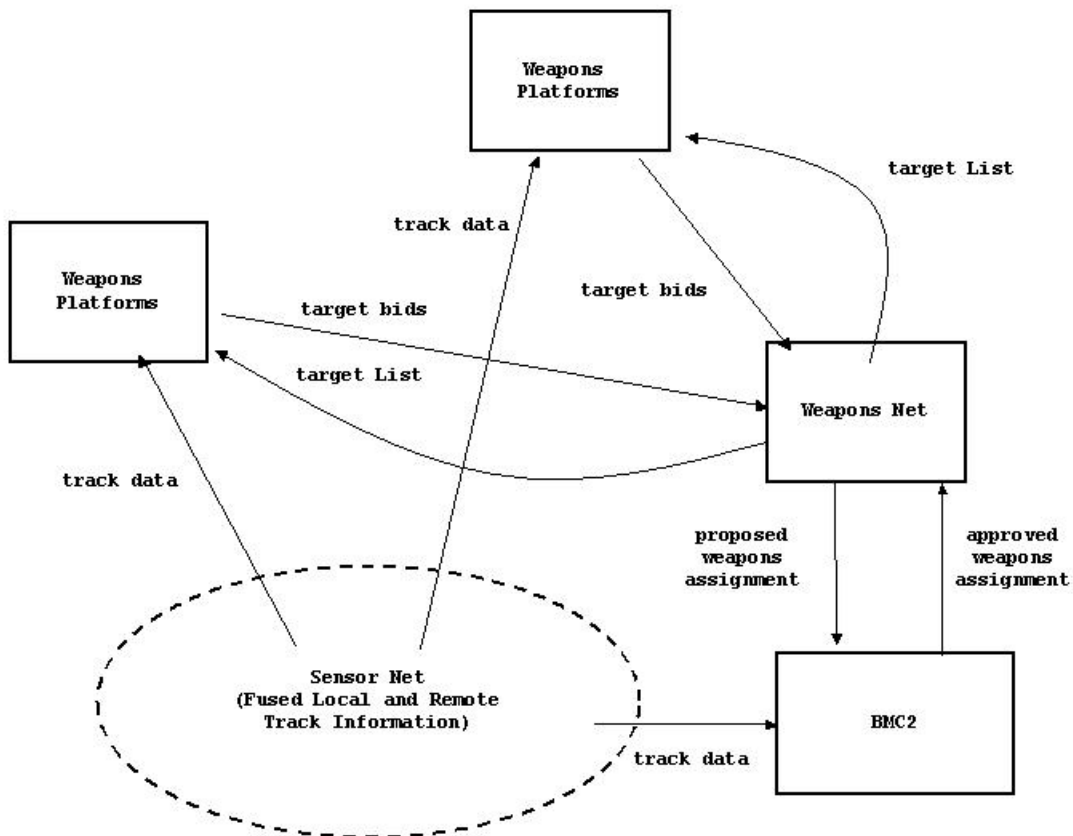


Figure 10. Use Case 3 Diagram

Context of Use: The goal of this use case is to assign targets to weapons via cooperative target bidding.

Primary Actors: Sensor Net, Weapons Net, Weapons Platform, BMC2

Stakeholders and Interests: Regional Commanders, Higher Commanders

Preconditions: Weapons Net is functional.

Success Guarantee: Weapon assignments are made.

Trigger: Weapons Net received a target list from the BMC2.

Main Success Scenario:

Weapons Net creates "target bidding request" for each target on the target list and broadcasts the information to all Weapons Platforms in the region.

Individual Weapons Platform examines the "target bidding request" data and the attached track data, matches its capabilities against the targets, formulates target bids and forwards them to the Weapons Net.

Weapons Net closes the bidding process for each target when each target's bidding time expires. Throughout the process this data is forwarded to the BMC2, which uses a target-bidding algorithm to create a weapons assignment. The BMC2 creates a weapon assignment message and replies to the Weapons Net.

The Weapons Net broadcasts the weapons assignment to all Weapons Platforms in the region.

Extensions:

1a: If Weapons Net fails to create target-bidding-request information, then the process fails.

3a: If a target does not receive a winning-weapon bid, Weapons Net will notify BMC2.

4a: If Weapons Net does not receive any acknowledgment (positive or negative) from the BMC2 after a predefined approval-time window, the Weapons Net will assume that the weapon assignment is approved by default.

Technical and Data Variations List:

Target-bidding-request information will include the target-track identification, extrapolated track information, time window for bidding, and any restrictions on the type of weapons used against the target.

A target bid will include the weapon identification, the intended target-track identification, proposed time to commence engagement, estimated time to intercept the target, and probability of kill success.

Weapons assignment information will include weapon and intended target identifications, estimated probability of kill, and earliest and latest time to time to commence engagement.

5. Use Case 4: Engage Targets

Context Diagram:

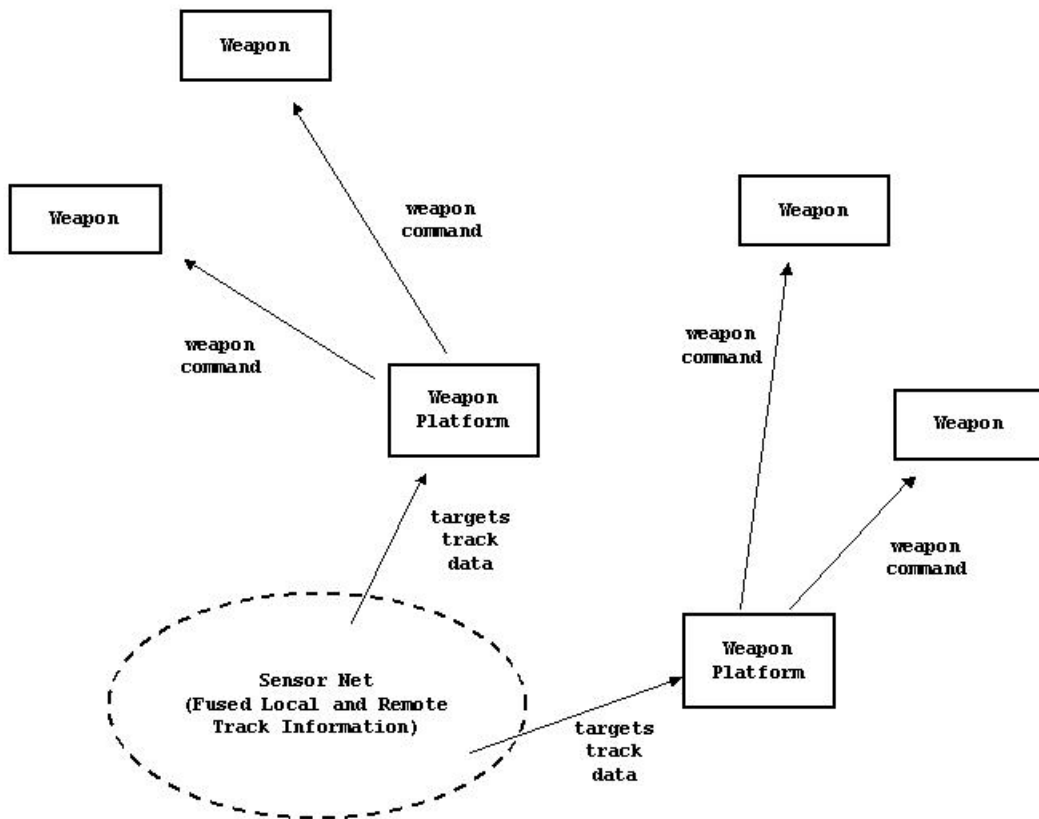


Figure 11. Use Case 4 Diagram

Context of Use: The goal of this use case is to engage threat ballistic missile.

Primary Actors: Sensor Net, Weapon Platform, Weapons, Interceptors

Stakeholders and Interests: Regional Commanders, Higher Commanders

Preconditions: Weapon Platforms and Weapons are functional.

Success Guarantee: Weapon successfully intercepts target.

Trigger: Weapon is assigned to engage a target.

Main Success Scenario:

Weapon Platform contacts Sensor Net and receives track information to develop a firing solution for its weapon.

Weapon Platform continues to update its firing solution using the track information from Sensor Net.

Weapon activates its interceptor within the interval defined by the earliest and the latest time to commence engagement.

Interceptor engages threat ballistic missile.

Extensions:

1a: If the assigned Weapon Platform fails to generate a firing solution, the Weapon Platform notifies Weapons Net and the target is re-bid.

3a: If the Weapon Platform receives an order from the BMC2 to cancel the weapon engagement before the weapon activates its interceptor, it will stand down the weapon and send a compliance acknowledgment to the BMC2. The BMC2 advises Weapons Net of the change of mission.

3b: If the Weapon Platform receives an order from the BMC2 to cancel the weapon engagement after the weapon is outside of the control of the Weapon Platform, it will send negative acknowledgment to the BMC2.

4a: If the interceptors fail to engage the threat ballistic missile, the process fails.

Technical and Data Variations List: none

6. Use Case 5: Assess Kill

Context Diagram:

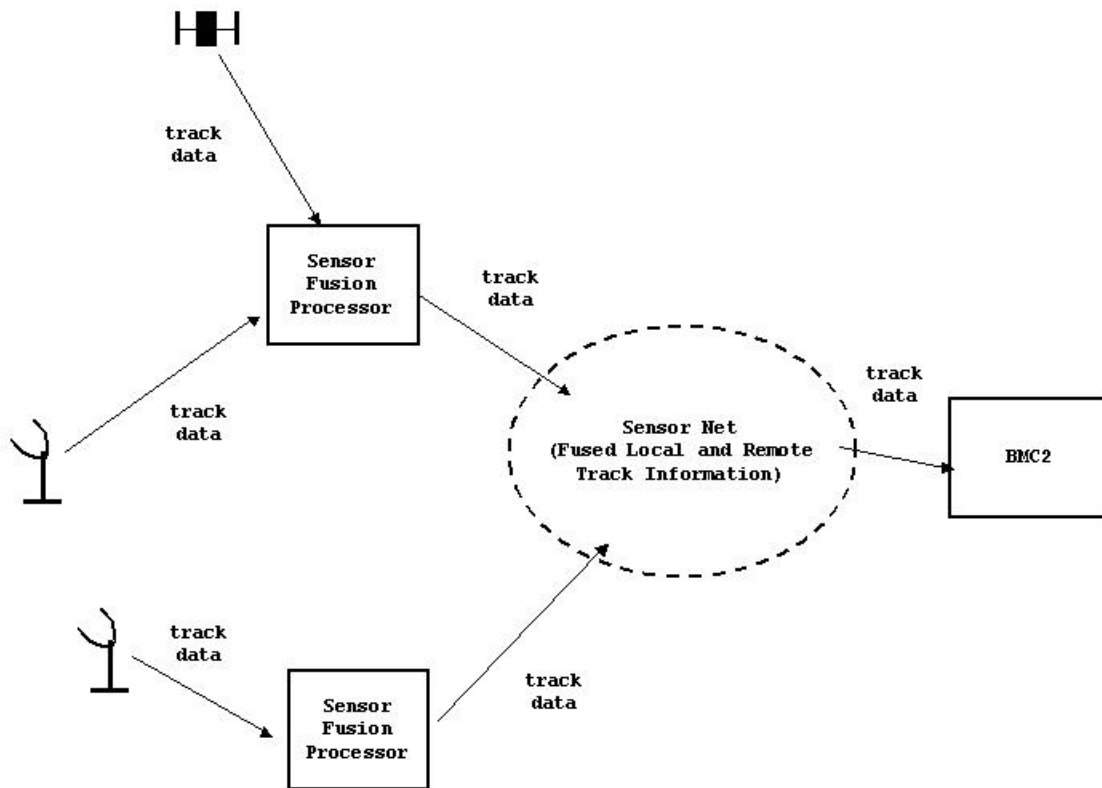


Figure 12. Use Case 5 Diagram

Context of Use: The goal of this use case is to determine the kill status of the threat ballistic missile.

Primary Actors: Sensor Net, Sensor Fusion Processors, Sensors, BMC2, threat ballistic missile

Stakeholders and Interests: Regional Commanders, Higher Commanders

Preconditions: Sensors, Sensor Fusion Processor and Sensor Net are all functional.

Success Guarantee: BMC2 determines that threat ballistic missile is destroyed and reports kill.

Trigger: Weapon engaged target.

Main Success Scenario:

Sensor Fusion Processors continue to identify and type-classify the threat ballistic missile events as shown in use case no. 2. It applies feature recognition processes, discriminates objects in debris clouds, and compares tracked objects to intelligence profiles.

BMC2's Kill Assessment Unit monitors and compares tracking data from Sensor Net for evidence of destroyed targets, and issues immediate probability of kill.

BMC2 determines that threat ballistic missile is negated and issues kill-assessment report.

Extensions:

1a: No Sensor Fusion Processor can discriminate objects. Organic weapon sensor searches debris cloud and discriminates objects and updates Sensor Net. If organic weapon sensors are unable to provide an update, the process fails.

2a: BMC2, based on data supplied by the Sensor Net, cannot determine with high enough probability that threat ballistic missile is negated. Sensor Net continues to carry track as active threat.

Technical and Data Variations List: none

E. CLASS DIAGRAM

Now we turn to refining the use cases into high-level abstract Class Diagram (Figure 13). The new classes Sensor Fusion Processor, Sensor Net, and Weapons Net are realized and the necessary messages and data are identified and included. The basic class framework and data-only interface strategy is retained to reduce coupling between components and realize the properties as defined in previous work.³⁶

F. SYSTEM SEQUENCE DIAGRAMS (SSD)

In order to further identify and refine the requirements, behavior, and timing constraints of the system based on the developed use cases and BMDS architecture we will utilize System Sequence Diagrams (SSD). The purpose of a SSD is to show the dynamic interaction of objects within a system by graphically depicting the time ordering of message passing among the objects. For each of the use cases described earlier, an equivalent SSD is given and are described in detail in Appendix D.

³⁶ Dale Scott Caffall, "Conceptual Framework Approach for System-of-Systems Software Developments" (M.S. Thesis, Naval Postgraduate School, March 2003), p 37.

THIS PAGE INTENTIONALLY LEFT BLANK

V. BMDS MODEL

A. INTRODUCTION

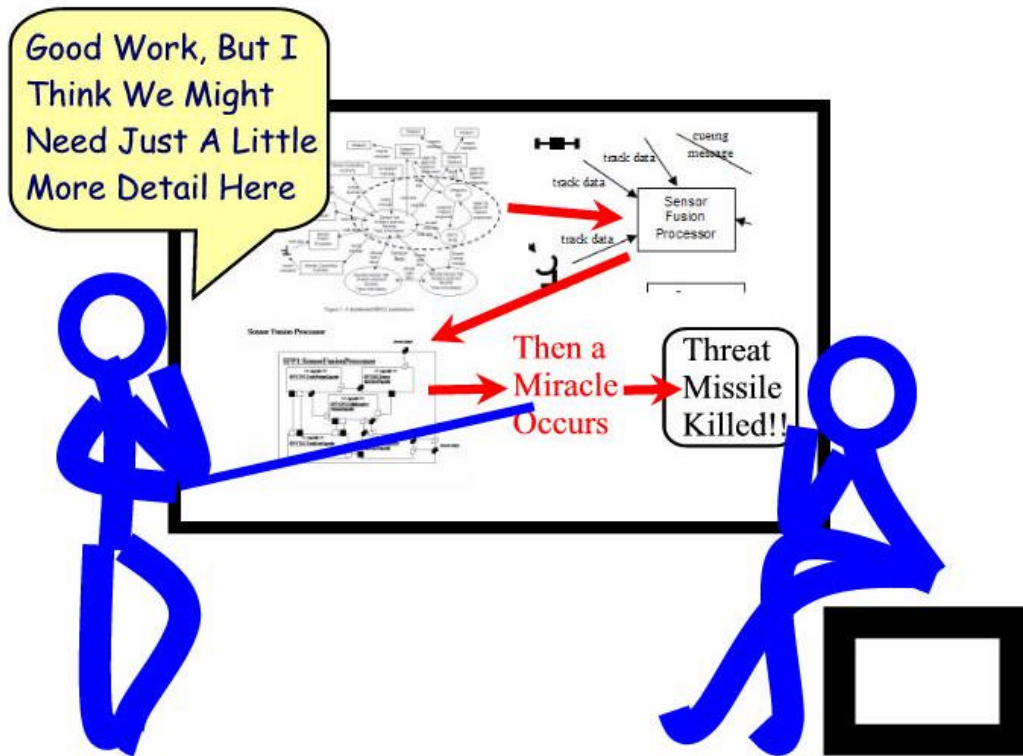


Figure 14. The Miracle

In deciding how to best transform from the abstract requirements to the concrete specification, we evaluated several technologies. The Unified Modeling Language (UML), while providing an adequate platform for this subject, was found to be inadequate for complex systems with real-time constraints. The Prototype System Description Language (PSDL), while a strong candidate for its ability to handle the real-time aspect of the project, we found to be not suitable for this particular task, as it is too fine grained and is only capable of modeling static systems (our models have dynamic modules). Therefore, in our search for a rough-order modeling language, we selected UML-RT, the

real-time extension for UML as specified in Selic and Rumbaugh "Using UML for Modeling Complex Real Time Systems."³⁷

In using UML-RT to model the specifics of the system, we followed a hierarchy plus input output process (HIPO)³⁸ approach. We modeled all nine major components from the use cases (Sensor, Sensor Controlling Authority, Competent Authority, Sensor Fusion Processor, Sensor Net, Weapon Platform, BMC2, Weapon, and Weapon Net). As our thesis focuses on the sensor portion of the Sensor-to-Shooter equation, only the Sensor Fusion Processor and Sensor Net are decomposed down to two levels of detail. Their models include Interface Capsules on the first level of decomposition, which shows the point at which they interface with peer assemblages. The second level of decomposition for these two components has Communications Capsules for communicating between capsules as opposed to components. This helps further decompose the complexity of each component. As the sensor portion of the use cases was our focus, the other six assemblages are not decomposed further and have implicit Interface Capsules, which are not explicitly shown on their first level of decomposition.

All of these models are included in this thesis in Appendix E. Please refer to Appendix E for a complete treatment of each model.

UML-RT is hierarchical and, as such, is excellent for decomposing complex systems into less complex pieces. If one examines Figure 6, an entity called SFP (Sensor Fusion Processor) can be found.

³⁷ Bran Selic and Jim Rumbaugh, *Using UML for Modeling Complex Real Time Systems*, Apr. 1998.

³⁸ IBM Corporation, *HIPO: A design Aid and Documentation Technique*, 1974.

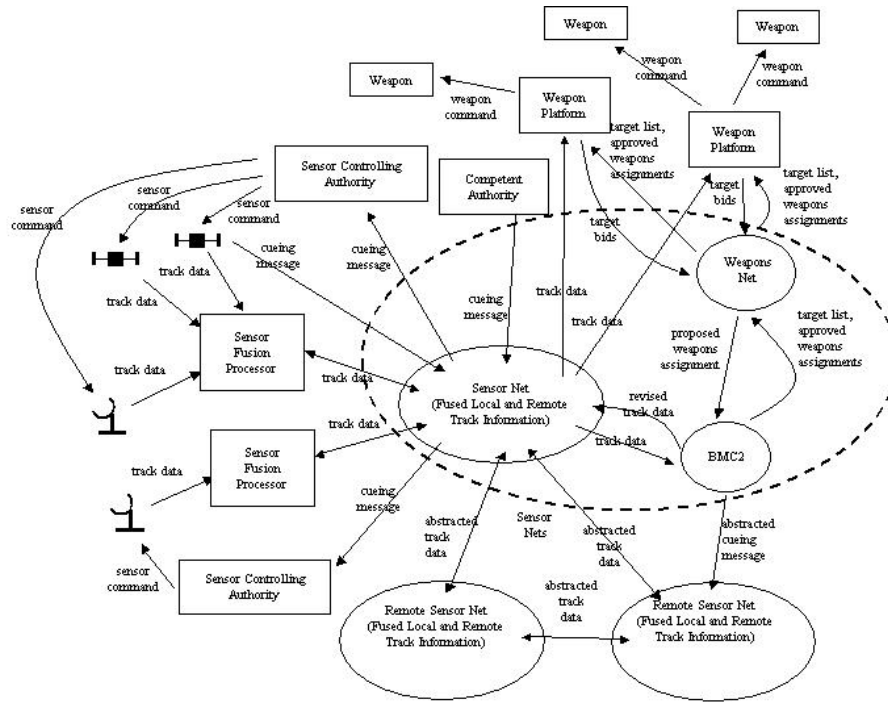


Figure 6.

Through use case analysis, we determine this entity to be necessary, and we determine which functions it should execute. The model can then be broken down into subcapsules, five of them in this case. Each of these subcapsules then is responsible for a piece of that functionality. As one decomposes, each subcapsule is treated as a black box, with input and output. Once each subcapsule is decomposed, however, the opaque blackbox becomes a transparent whitebox, where the processes are either described by state machines or subcapsules that can be further decomposed.

Sensor Fusion Processor

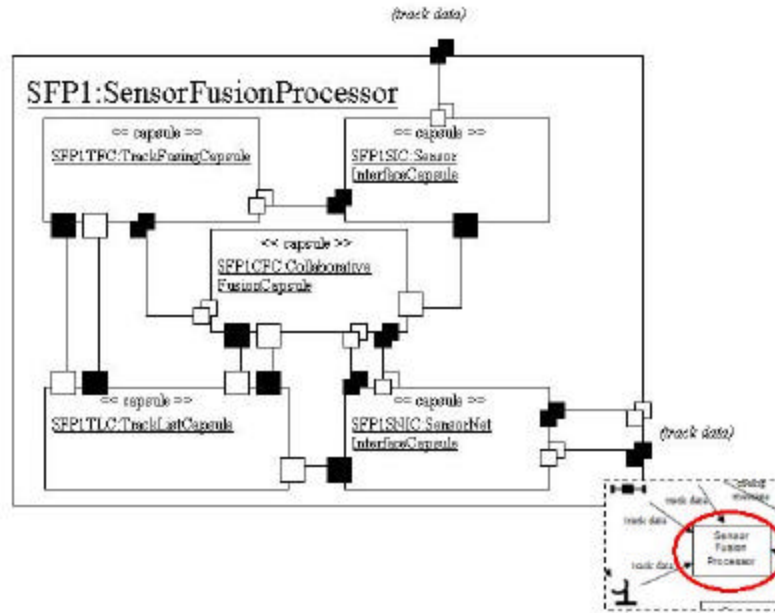
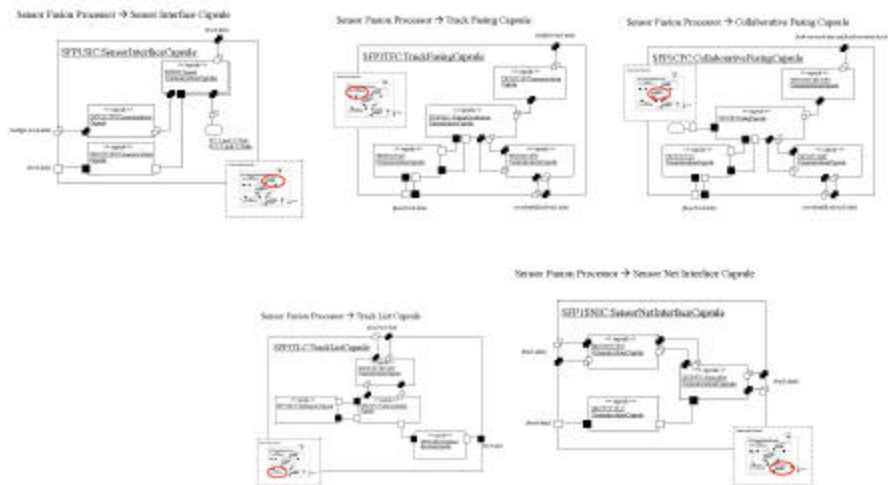


Figure 26

Each of those five sub-capsules is then further decomposed into multiple sub-capsules in the following five figures.



Figures 27 through 31.

Each sub-capsule also contains sub-capsules, which can be further decomposed if desired eventually down to a state

machine. This is the process we used to decompose a highly complex system into a moderately complex system, and which could be used to decompose it further into a simpler system.

One more point on vocabulary; the sensor portion of the entire Sensor-to-Shooter equation is strictly concerned with tracks. Anything which the sensors are tracking is a track. The Sensor Net maintains a master track list for all affiliated Sensor Fusion Processors, Weapon Platforms, and the BMC2 to reference. Tracks only become targets once the BMC2 designates them as targets by putting them on the master target list.

This HIPO decomposition has proven to be very useful in determining where the major functions should be, as well as for describing the highest-level algorithms (or processes) of the system. By breaking the system into capsules (smaller, less complex units) and sub capsules, with a few state machines, one begins to see where the most time-critical pieces of the system are and how they can be accommodated. Finally, by following this process, one begins to see how an inherently complex system can be broken down into more understandable pieces.

B. CONTEXT

The context of the UML-RT model is based directly on the BMDS architecture (Figure 6) and is annotated as a sub-graphic on each functional depiction shown with the dotted line on a box. As decomposition occurs the graphics will display the higher level that invokes it.

C. ASSESSMENT

We learned several things about the system by doing this decomposition.

- Passing firing solution quality data to a weapon platform that has won a target bid is the highest priority task. Therefore, it has been streamlined to ensure the fastest, best response possible.

- We realize that it would be difficult for us to put real-time requirements on every aspect of the system. As such, we modeled the passing of firing-solution-quality data to a weapon platform in a small number of capsules to help lower the complexity and, therefore, increase the capability of the system to be specified using hard-time requirements.

- Cueing for potential targets is the second highest priority task. Therefore it is also streamlined.

- In deciding how to handle cues, it became obvious that, since potentially tens of platforms could be simultaneously sending cueing messages, all of them directly to the Sensor Net, we had to add into the model a simple XOR-style correlation capability.

- It was our opinion that the master track list maintained by the Sensor Net should be the source to develop the 'common operating picture' used by all Sensor Fusion Processors. Therefore, we modeled it as being pushed out to all SFPs, with each SFP maintaining a materialized view (i.e., maintaining a local copy and temporarily maintaining local differences from the master track list). These local changes start out as pending updates submitted to the Sensor Net for incorporation, or

if rejected, then they become local data. One example of local data is, if the SFP has a lesser quality track than that which is already on the master track list, it would hold its track until it beat the one listed on the master target list or until the track either disappeared or was determined to be a different track from any listed on the master track list.

•At the time of completion of this thesis, there is still quite a debate out there on what constitutes 'sensor fusion'. Therefore, for the sake of consistency in this chapter and Appendix E, we provide the following definitions:

•Track Correlation: Comparing two tracks to determine if they are the same actual object or two different objects, resulting in one track being forwarded and the other being dropped.

•Track Discrimination: Filtering out objects that are not relevant (debris, chaff, decoys) in order to reduce the load on the system/network.

•Track Fusing: The act of taking a track and filling in any gaps in coverage by using a different track taken from a different sensor/perspective, or of taking a track taken from a different type of sensor and using it to enhance the base track, making the ellipse of certainty (the area where we think the missile is) smaller and more precise.

•Track Abstraction: The act of sampling a real-time track, making it less accurate but lighter in data load and therefore easier to pass, to give larger granularity situational awareness to peer Sensor Nets. This is useful between Sensor Nets to broaden situational awareness beyond

one's own local area. It is also the method used to pass cues to Sensor Controlling Authorities.

VI. BMDS OMNET ++ SENSOR FUSION PROCESSOR (SFP) SIMULATION

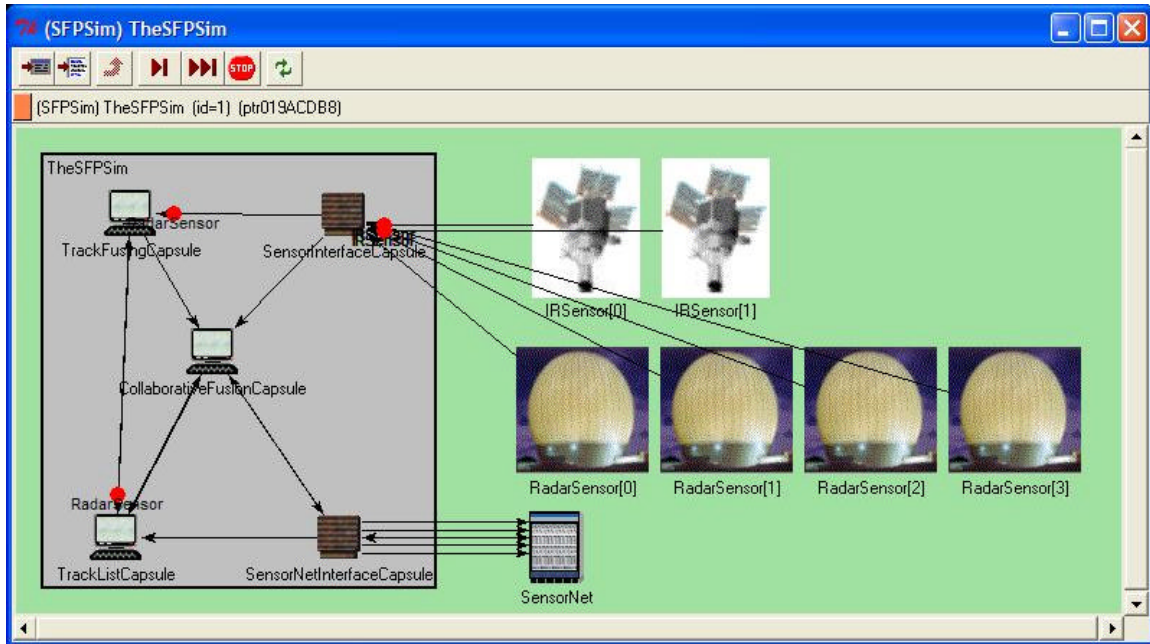


Figure 15. OMNet++ BMDS SFP Simulation.

In deciding on a simulation tool for modeling the Sensor Fusion Processor, we evaluated two well-known general-purpose simulation systems; MATLAB and OMNeT++. We found that MATLAB is too fine grained of a simulator than is needed at this stage of development, and that OMNeT++ better fits our requirements for a coarse-grain level of modeling fidelity. We developed a simulation based on the device which is least understood in the entire system -- the Sensor Fusion Processor (Figure 15) -- in order to further our understanding of how it would function within the context of the Ballistic Missile Defense System.

The design for the simulation flowed naturally from the UML-RT model, which is included in the appendices of this thesis. We found a one-to-one correspondence between our model and the OMNeT++ sub-modules, gates, and parameters.

The following are significant assumptions that we made while developing our simulation models of the BMDS:

- Modules within the SFP are collocated (included parameter is Gig-E transmission speed between modules).
- Message sizes are based on estimates of what data would be required from each device.
- There are many proposed algorithms proposed for sensor fusion. The inputs, outputs, and time delays represented in our simulation are representative of those algorithms. These parameters can also be easily replaced with more specific parameters as needed.
- Discrimination occurs at the individual sensor and is not explicitly part of the simulation, as the SFP is not designed to screen out debris, decoys, etc.

Our input parameter values are derived from the following:

- Number of Ground-Based Radar Sensors: Typical of a standard theater defense.
- Number of Satellite-Based IR Sensors: Typical of the number of satellites involved in a standard theater defense.
- Number of actual objects being tracked: Arbitrary, but based on typical world scenarios.

- Data Rate (bps) between SFP and Sensor Net: Default rate is based on CPT Joel Babbitt's 3.5 years of strategic communications experience.
- Data Rate (bps) between Capsules: Based on Gigabit Ethernet.
- Size (bits) of Fused Track: We based it on one of two assumptions, being that fusion only replaces pieces of a track rather than adding to it, or that a track increases in size when fused.
- Data Rate (bps) between Radar Sensor and the SFP: Based on current telecommunications standards.
- Size (bits) of an Unfused Radar Track: Based on existing military systems.
- Delay (sec) between Radar Tracks sent to the SFP: Based on existing systems.
- Data Rate (bps) between IR Sensor and the SFP: Based on distance from the earth, downlink frequency, and speed of light.
- Delay (sec) between IR tracks being sent to the SFP: Based on rough order parameters of current systems.
- Size (bits) of an unfused IR track: Based on existing systems.
- Delay (sec) between Master Track List broadcasts: Based on how often it would have to be done in order to ensure target accuracy and aid collaborative fusion.
- Number of collaborative fusion requests from CFC: Arbitrary, but must be equal to or less than the

number of actual objects being tracked, as you cannot fuse more tracks than you actually have.

- Time (sec) each Module takes to handle a track:
Based on an estimate from a Professor Wen Su assuming level 3 Internet Protocol routing for each discreet message in the simulation and top of the line 2003 hardware and routing software.
- Time (sec) to check a track against the List:
Based on an estimate from Professor Wen Su assuming associative memory, data stores (as opposed to data bases), and the top of the line 2003 hardware and memory management software.
- Time (sec) required to perform a Fusing Action:
Arbitrarily set, as this function is still not well defined by MDA.

The parameters all have ranges and interdependencies discussed in the analysis of the results of the simulation done in Chapter VII of this thesis and more fully analyzed in Appendix G. The data types for each parameter were set in the body of the simulation code. All parameters, which take seconds, are doubles (thereby allowing milliseconds, microseconds, etc.). All parameters, which take bits or bits per second, are integers (as there is no such thing as a 'partial' bit).

As a caveat, we have not fully validated this model over the entire range of possible realistic values. The user should establish the validity of the outputs generated by exercising the model over the range of values of interest.

Knowing that there are many who may come after us, we designed this simulation to be extensible. Parameters,

processes and algorithms are commented and explained in the body of the code (See Appendix F).

THIS PAGE INTENTIONALLY LEFT BLANK

VII. DISCUSION OF RESULTS

The most significant timing constraint placed on the BMDS is the need to destroy a threat ballistic missile before it exceeds the weapons-system capabilities for a successful intercept. This assessment must be applied through each phase of flight of the threat ballistic missile for each type of weapon that it has the potential to engage. For instance, the duration of the boost phase is between one to four minutes depending on the type of missile. In this case, we say that the threat missile is an ICBM that will accelerate to Mach 9 (though not until exoatmospheric) and will be in the boost phase for the entire four minutes and that the interceptor, which will accelerate to Mach 4 (at a much faster rate then the ICBM) can conduct the intercept within this phase but not beyond into the exoatmospheric region. This means that the entire kill chain of events leading to destruction, and the messaging between the BMDS objects as depicted in the SSD, must occur within this timeframe or else the intercept will not be physically realizable.

Working the problem from the projected point and time of expected intercept, the HTK missile must be "off the rail" with a sufficient amount of time to track and hit the target. Therefore, the time of flight necessary for the interceptor must be subtracted from the total time available. If the interceptor requires (at maximum range) one minute to intercept the ICBM, then the latest possible time that the interceptor can be launched is at three minutes from the time of the ICBM launch. If there is weather precluding observation by space-based IR sensors or physical obstructions to GBR's such as mountainous terrain,

the time for initial detection is increased and overall reaction time is decreased. In the case of weather obscuration, detection could be delayed for up to one minute until the threat missile penetrates and is above the cloud layer for IR sensor detection; this would require the BMDS to conduct all of the required tasks within two rather than three minutes. Therefore, the BMDS must be able to detect, track, assign, and engage the ICBM, and perform all the necessary communications within that same timeframe.

Looking at the kill chain from initial detection, the time it takes a sensor to receive enough hits to develop a track and forward that track out to a SFP will vary based on the distance of the target from the sensor, the update rate and number of hits required to develop a track for each specific sensor. The time necessary to transmit a signal and receive a contact hit from a target is twice the distance divided by the speed of light. The general function for developing a solid track is based on applying a fast Fourier transform (FFT) algorithm, the calculations of which are on the order of $n \log n$, where n is the number of sampled values in a particular range bin.³⁹

Once a track is developed it must be compared against a database of tracks local to the sensor to either update an existing track or develop a new track. Tracking the target and applying gates have a time complexity on the order of $nm \log nm$, where n is the number of established trajectories and m is the number of measured values of targets that n can be mapped to.⁴⁰ All of the calculations can be conducted in a timely manner and are within the

³⁹ Jane Liu, *Real Time Systems*, Prentice Hall, Upper Saddle River, N.J., 2000, p. 16.

⁴⁰ Ibid, p. 19.

realm of high-speed processors and are further enhanced by electronically steered phased-array radars.

Once the track has been modified it must then be transmitted to a SFP for further processing. Our simulation model incorporates an update rate of 0.5 seconds for an active sensor. This number was selected based on the fact that we have abstracted out the specific type of sensor. Passive sensors are typically updated at a lower rate based on the need to observe the target longer, requiring more hits to develop a track as opposed to active sensors. This is the case with space-based IR satellites, because both the detection distances are greater and there is a requirement to sample more hits in order to refine a passive track; our simulation model uses a delay of two seconds to model the initial track development and transmission.

Distance, with regards to communication, will also come into play as a significant factor affecting timing. Satellites in geosynchronous orbit require a minimum of one thirteenth of a second for a transmitted signal to travel between the satellite and the earth receiving station; this is just a rough estimate based on the distance divided by the speed of light and does not take into consideration the duration of the transmission nor the amount of information to be carried on the frequency. Terrestrial units, while having greater bandwidth, must also traverse long distances, in some cases up to half the circumference of the Earth. Sea-based units potentially will experience the greatest time delay with communications. In contrast to line-of-sight communications, Naval surface combatants rely on satellite-based communication; such communication requires multiple paths and transmissions to participate in the BMDS.

After a track is developed locally, it is sent to a SFP to be further processed by fusing multiple sensor tracks into a single track for distribution to other units. These tracks will be arriving at an asymmetric rate based on the update and transmission of each of the different sensors. Discrimination, filter, and fusing algorithms processed by the SFP are applied to all received tracks. The timing requirements imposed by these algorithms cannot be assessed as the algorithms are to be developed.

The summation of all the delays in transmitting and receiving data must be incorporated within the simulation to provide a realistic representation of timing within the system.

When looking at the BMDS and the messaging that must occur, as depicted in the SSD's, there are multiple instances where potential bottlenecks could occur.

The potential is first identified at the SFP with numerous sensors providing track data at asymmetric rates. The SFP must discriminate, filter, and fuse the current track data into a single track and forward that track out to the Sensor Net. The SFP must also compare that track with a track database to determine if it is to be used to update an existing track or develop a new track. Additionally, developing a rough track classification requires a table look up, data comparison, and association based on known parametric data.

The Sensor Net, once it receives the tracks, distributes them to participating BMC2 elements, which must then evaluate and determine which tracks warrant classification as threat ballistic missiles, developing a master target list to prioritize those targets based on Impact Predicted Points (IPP), priority defended areas, Rules of Engagement (ROE), etc., and forwarding that list

to weapons-capable platforms to evaluate their capability against the threat. This process consumes time that must be accounted for within the total available time to conduct a successful intercept.

A parametric (i.e., sensitivity) analysis involving numerous simulation runs was conducted to determine what were the most significant timing constraints on the system and at what point they became critical, which are annotated in Appendix G with both data tables and line graphs. A methodical approach was utilized in the process of obtaining data where one input value was varied and the others remained constant to see how that one variable impacted the system. For data input values, we utilized commercial data-transmission rates and approximate system clock-speed values for internal timing. In doing so we abstracted the data points and precluded any implication of existing or developmental systems, while still obtaining valid research data.

The most significant timing issue that was obtained through multiple iterations of the simulation was that as the track load increased, whether it was from large numbers of tracks being reported, moderate numbers of tracks being reported by large numbers of sensors, or when the sensors increased their update rates, the Track List Capsule would become saturated, thus increasing the time to transmit track data. This is evidenced by corresponding increases of both the TLC percent utilization and average time to broadcast tracks on the Sensor Net with an increase in the number of tracks reported or sensors updating.

We observed that track message size and data throughput rates had little impact on the time to transmit track data. Additionally, as the number of track collaboration requests increased, the number of normal

tracks dropped to zero and collaboratively fused tracks increased, but there was no real impact to the overall average track-process time. As to be expected, increases in module-processing time, track-list-comparison times, and track-fusion times all had corresponding increases to the average track processing and throughput values.

Having now finished one full pass through the Use Case-Model-Simulation cycle, we found that the Use Cases feed directly into the UML-RT models, which in turn flow directly into the OMNeT++ simulation. In addition, issues that arise in building, running, and analyzing the simulation and simulation outputs provide direct feedback on the validity of the model, which in turn provides direct feedback to the validity of the use cases themselves.

Following this feedback loop ourselves, we found it necessary to redesign the SFP's Track List Capsule in order to handle heavier work loads, in this case more traffic, as shown in Figure 16. Based on the results of the simulation with the redesigned Track List Capsule included, we conclude that the redesign is not sufficient, and that a redesign of the SFP as a whole, which would probably place a slave TLC off of the master TLC local to each of the fusing capsules, would be required to reduce the load on the Track List Capsule and prevent the system from bottlenecking there.

Sensor Fusion Processor → Track List Capsule

SIMULATION REDESIGN

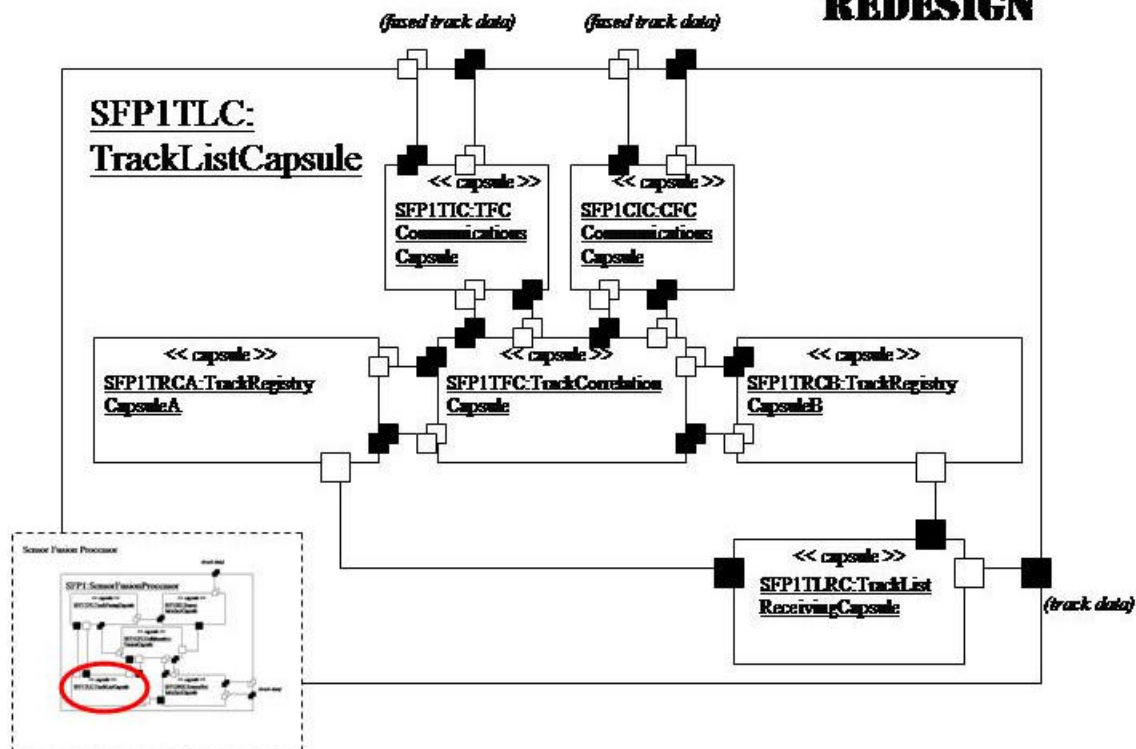


Figure 16. SFP Track List Capsule Redesign

As a redefinition of this capsule's sub-capsules, we specified the changes listed below:

- **Track Registry Capsules (A and B):** Maintains the SFP's master list of all perceived valid tracks as well as any additional tracks received from the Sensor Net, including any commands added to received tracks or commands pertaining to the locally maintained tracks. Only Track Registry Capsule is active at a time. The other is in a semi-active state, in which it is receiving all updates from the Track Correlation Capsule, but it is not being used by the Track Correlation Capsule to serve TFC/CFC correlation requests. However, when it receives a newer copy of the master track list from the Track List Receiving

Capsule than that which is held by its active counterpart, it goes active and directs the other active capsule to go into semi-active mode.

- **Track List Receiving Capsule:** Receives the Track List sent out periodically from the (higher level) Sensor Net. It sends the Track List to the semi-active Track Registry Capsule first, then, after the active becomes semi-active, it forwards the list to the newly semi-active capsule.

VIII. CONCLUSION

A. SUMMARY.

In a field of study that is not well defined such as ballistic missile defense and which consists of systems of systems, one must discover and develop methodologies for refining requirements and ensuring a project's purpose is successfully accomplished. The Use Case-Model-Simulation feedback cycle that we used is a systematic engineering methodology for developing such highly complex systems of systems.

Through the use of this methodology, we were able to establish a feedback cycle. Using this cycle, we were able to find weak points in the Sensor Fusion Processor, which we then redesigned and validated through simulation. This type of feedback or refinement loop is key to ensuring success in distributed software development in a Software Engineering Environment (SEE).

B. RECOMMENDATIONS.

There are several areas of future study, some of which are listed here:

- Validation of the model, to include running test cases for likely scenarios.
- Expanding or enlarging the model by representing the rest of the Ballistic Missile Defense System.
- Inclusion of specific algorithms and specification of interfaces for future inclusion of algorithms.
- Integrating this methodology with a SEE tool such as Rational UDX.
- Documenting requirements traceability and analysis which follows the requirements from Use

Cases to the model and finally to the simulation artifacts.

- Application and enhancement of MDA's simulation models which collectively form the BMDS Core Model set, focusing primarily on the BMD System Level M&S (Modeling and Simulation).⁴¹

⁴¹ Kevin J. Greaney, "Evolving A Simulation Model Product Line Software Architecture From Heterogeneous Model Representations" (Ph.D. dissertation, Naval Postgraduate School, Sept. 2003), p. 224.

APPENDIX A. GLOSSARY

ABL. 1. Airborne Laser. 2. Aircraft Based Laser.

ABM. Anti-Ballistic Missile.

ABM Treaty. Anti-Ballistic Missile Treaty of 1972, signed and ratified by the (former) Soviet Union and the United States, limiting deployment on each side to one site comprising 100 interceptors, 100 launchers, and several ground-based radars. The Treaty also regulates development and testing. In December 2001, President George W. Bush announced that the United States would withdraw from the treaty, which the U.S. did in June 2002

Acquire. 1. When applied to acquisition radars, to detect the presence and location of a target in sufficient detail to permit identification. 2. When applied to tracking radars, to position radar beam so that a target is in that beam to permit the effective employment of weapons.

Acquisition (ACQ). (Sensor) The results of processing sensor measurements to produce object reports of interest to the system.

Active. In surveillance, an adjective applied to actions or equipment, which emit energy capable of being detected, e.g., radar is an active sensor.

Active Defense (TBMD). Active defense protects against theater missiles by destroying them in flight. Engagement capability is required throughout all phases of the missile's trajectory (boost, post-boost, mid-course, and terminal) to prevent saturation of point defense, to negate warhead effects, and to ensure minimal leakage in defending critical assets. Therefore, active defenses must consist of defense in depth to provide multiple engagement opportunities with differing technologies, increasing the probability of kill, and countering the enemy's counter-measure efforts. Active defenses could consist of space-, air-, ground-, and sea-based systems. If a strategic ballistic missile defense system is deployed, the active TMD should be supported by, but not limited by, those systems to increase the defense in the theater of operations. Active defense is considered one of the four pillars of TMD capability.

Active Homing Guidance. Guidance system in which both the source for illuminating the target, and the receiver for detecting the illuminating energy reflected from the target is carried within the missile.

Active Sensor. One that illuminates a target, producing return secondary radiation, which is then detected to track and/or identify the target. An example is radar.

AEGIS. A totally integrated shipboard weapon system that combines computers, radars, and missiles to provide a defense umbrella for surface shipping. The system is capable of automatically detecting, tracking, and destroying air-borne, sea-borne, and land-launched weapons.

AEGIS BMD. Aegis Ballistic Missile Defense (Aegis BMD) Project is an element of the Ballistic Missile Defense System, and is being developed to provide a rapidly deployable, highly mobile defensive system capability against short-to-intermediate range ballistic missile attacks on population centers, debarkation ports, coastal airports, amphibious objective areas, expeditionary forces, troops, friends, and allies. Forward positioning of the ship makes possible a missile defense that will protect vast areas, often-entire countries. The Aegis BMD element of the BMDS builds on the proven Mark 7 Aegis Weapon System including modifications to the Standard Missile, and the Mark 41 Guided Missile Launch System.

AEW. Airborne Early Warning.

Ballistic Missile (BM). A rocket-propelled vehicle moving under its own momentum and the force of gravity that does not rely upon aerodynamic surfaces to produce lift and consequently follows a ballistic trajectory when thrust is terminated.

Ballistic Missile Defense (BMD). All active and passive measures designed to detect, identify, track, and defeat attacking ballistic missiles (and entities), in both strategic and theater tactical roles, during any portion of their flight trajectory (boost, post-boost, midcourse, or terminal) or to nullify or reduce the effectiveness of such an attack.

Ballistic Missile Defense System (BMDS). 1. An integrated system of all BMD sensor and weapon systems. This system-of-systems will provide greater capabilities to defend against ballistic missile attacks. 2. The aggregate BMD BMC3 and BMD forces that, in total, provide defense against ballistic missile attacks to North America and other areas of vital interest.

Battle Management. Battle management is composed of two parts, namely, strategies and the actual collection of tasks to be performed to successfully implement chosen strategies. Examples of strategies are: area defense, adaptive preferential defense, offense deployment, and

rules of engagement depending on the evolution of battle, etc. Examples of tasks are, resource allocation, target assignment, probability of kill calculations and kill assessment, etc. Given a set of strategies, resources and hostile asset deployment, battle management addresses the problem of choosing a strategy or set of strategies and performs the associated tasks that would result in the most "desired" outcome.

Battle Management/Command and Control (BM/C2). The BM/C2 is the equipment, communications networks, and processes which the warfighting Combatant Commanders will use to monitor the theater ballistic missile fight and to direct the activities of the various BMDS elements.

BMC2. A set of computer workstations with software and communications gear providing full set of BMDS applications at a command center.

BMDS Block. The Missile Defense Agency (MDA) intends to field a set of software packages every two years. These sets of software packages are called BMDS Blocks.

BMDS Elements. These are the systems that as a single entity provide BMDS capability.

Boost Phase. The first phase of a ballistic missile trajectory during which it is being powered by its engines. During this phase, which usually lasts 3 to 5 minutes for an ICBM, the missile reaches an altitude of about 200 km whereupon powered flight ends and the missile begins to dispense its reentry vehicles. The other phases of missile flight, including midcourse and terminal, take up to the remainder of an ICBM's flight time of 25 to 30 minutes.

Boost Defense Segment (BDS). The portion of the BMDS that defeats ballistic missiles in the period of flight prior to the termination of powered flight.

Command. Authorization required to perform command operations for command-oriented functions.

Command, Control, Battle Management, and Communications (C2BMC). An independent command and control capability from which ballistic missile defense operations can be implemented. The C2BMC allows for full situational awareness and devolution of command (if necessary). Each C2BMC Node will have the capability of planning, coordinating, directing, and controlling surveillance and engagement operations.

Command, Control, Communications, Computers, and Intelligence (C4I). Procedures and technologies supporting

command and control, communications, and intelligence requirements.

Cooperative Engagement. Engagement of a target through cooperative use of resources and/or data from more than just one participating unit.

Cooperative Engagement Capability (CEC). The capability to engage a target through cooperative use of resources and/or data from more than one participating sensor. There are several forms of cooperative engagement, including 1. use of a composite track to launch a defensive weapon against a target, 2. fire control guidance of an interceptor using a composite track, and 3. near-real-time shift of interceptor control from one firing unit to another to improve overall defense system or architecture performance.

Correlation. The process of assigning or computing weights to determine that two or more tracks, consisting of smooth state estimates and representation of the uncertainty of the estimates, are for the same object or that they are for separate objects, or the result of that process.

Cued Operation. The directing of one sensor based upon the data received from another sensor.

Cueing Command. The command within a tactic, which specifies the sensor element's coverage volume.

Cueing Data. Cueing data is a subset of object tracks within a sensor element's coverage volume.

Data Fusion. Multilevel, multifaceted process dealing with automatic detection, association, correlation, discrimination, situation awareness, and threat assessment by combining data and information from single and multiple sources.

Data Link. 1. Means of connecting one location to another to transmit and receive data. 2. Particular path between two nodes over which data are transmitted. It includes transmission medium and digital-to-analog converters, modems, transmission equipment, antennas, etc., associated with this path.

Defended Asset List (DAL). A ranked listing of facilities, forces, and national political items that require protection from attack or hostile surveillance. The list is compiled from Federal departments and agencies, Unified and Specified Commands, and the Armed Services to ensure National Security Emergency Preparedness functions.

Defense Support Program (DSP). System of satellites in geo-stationary orbits, fixed and mobile ground processing

stations, one multi-purpose facility, and a ground communications network. Primary mission is to provide tactical warning and limited attack assessment of a ballistic missile attack.

Detection. Discrimination of an object from its background and its assignment to the class of potentially interesting objects.

Distributed Bidding. A distributed and automated process that (1) implements rule sets to assign and communicate among participants the weights or scores (i.e., the bid) of a system/platform's ability to conduct an engagement or perform some discrete sensor support activity and (2) recommends the weapon system(s) or sensor(s) to execute the engagement or perform the discrete sensor support activity based on the distributed bids.

Early Warning. (1) Early detection of an enemy ballistic missile launch, usually by means of surveillance satellites and long range radar. (2) Early notification of the launch or approach of unknown weapons or weapon carriers.

Endo-atmospheric. Within the earth's atmosphere. The altitude commonly used to separate the endo- and exo-atmospheric regimes varies from 100 km to 120 km.

Engage. In air or missile defense, a fire control order used to direct or authorize units and/or weapon systems to fire on a designated target

Engage On Remote (EOR). An advanced engagement operation where track data from external sensor(s), in the absence of local sensor data, are passed to the fire control component of a weapon system which uses these data to calculate launch parameters, fire the interceptor, and provide in-flight target updates to the interceptor. The local weapon command center retains control and responsibility for the engagement.

Exo-atmospheric. Above the atmosphere where the drag is negligible. The altitude commonly used to separate the endo- and exo-atmospheric regimes varies from 100 km to 120 km.

External Sensor. Sensor program external to Missile Defense Agency, e.g., national assets and service sensors.

Family-Of-Systems. A set or arrangement of independent systems that can be arranged or interconnected in various ways to provide different capabilities. The mix of systems can be tailored to provide desired capabilities dependent on the situation

Field of View (FOV). The angular measure of the volume of space within which the system can respond to the presence of a target.

Fire-Control Quality Data. The system-specific data required for a weapon system to compute a fire-control solution and conduct an engagement.

Forward-Based Sensor. Sensor deployed close to target launch point.

Forward Pass. The act of passing control and responsibility of an interceptor missile in flight from the launching command post to another command post. Sensor and/or guidance information may be generated by the launcher or may originate elsewhere. Interceptor tracking and in flight updates may continue to be performed by the launching unit, or from another guidance and control element, or a combination of the two.

Fusion. 1. The combining of automatically correlated information with data that refines the information or presents it in an intuitive format. Fused data in many cases will arrive later than real or near-real-time data. 2. Once associated or correlated, the process of combining all sources of information to improve the quality of the knowledge of the object. (e.g. a radar, an ECM intercept receiver, and a spotter all report on an object at location "X." In creating the track file on that object, all information is used to either improve the accuracy of location, or amplify on the nature of the object.

Gateway. 1. A gateway in a communications network is a network node equipped for interfacing with another network that uses different protocols. A gateway may contain devices such as protocol translators, impedance matching devices, rate converters, fault isolators, or signal translators as necessary to provide system interoperability. It also requires that mutually acceptable administrative procedures be established between the two networks. A protocol translation/mapping gateway interconnects networks with different network protocol technologies by performing the required protocol conversions. 2. A generic term for a C⁴I network node designed to provide interoperability by interfacing between two (or more) systems or networks that use different protocols. There are two types of gateways: (a) data forwarders between two or more tactical data links (TDLs), or between a TDL and a non-TDL system, and (b) routers and retransmitters (previously referred to as "cross-banding"). All gateways require the establishment of mutually

acceptable procedures for interfacing between the connected systems or networks.

Geo-stationary Orbit (GSO). An orbit 35,784 km above the equator. A satellite placed in such an orbit revolves around the earth once per day, maintaining the same position relative to the surface of the earth. It appears to be stationary, and is useful as a communications relay or as a surveillance post.

Global Positioning System (GPS). The Navstar Global Positioning System is a space-based radio navigation network providing precise positioning needs of all military Services. When fully operational, 18 satellites are in 6 orbital planes with an orbit period of 12 hours at 10,900 nautical miles altitude. Each satellite transmits three L-band pseudorandom noise-coded signals, one S-band, and one ultra high frequency for spacecraft-to-spacecraft data relay.

Gridlock. 1. The process of removing navigational and radar biases by calibrating to a common force reference point. This is accomplished by all units of the force simultaneously recording the position of a commonly held target that has a specified relative position from the force center (or other reference point) at the same instant. 2. The computer process used to compare an individual ship's track data with remotely originated track data, and to determine the correction necessary to bring the tracks into alignment.

Ground-Based Interceptor (GBI). A kinetic energy exoatmospheric interceptor with long flyout range to provide, where possible, a multiple engagement capability for defense of the U.S. with a relatively small number of missile launch locations. It is designed to engage post-boost vehicles and/or RVs in the midcourse phase of flight.

Ground-Based Radar (GBR). A task-able, modular, multi-function, phased-array radar that provides surveillance, tracking and engagement planning data in post-boost, midcourse, and terminal flight phases within its capabilities. It also provides target discrimination, in-flight target updates (IFTUs), and target object maps (TOMs) to interceptor vehicles.

Hand-Off. The transfer of a track file from one sensor or system to another system in which the first does not continue to track.

Handover. 1. The transfer of a track file from one sensor or system to another system in which the first sensor or

system continues to track the objects. 2. The successful acquisition of a target using data from the cue.

High Earth Orbit (HEO). An orbit about the earth at an altitude greater than 3,000 nautical miles (about 5,600 kilometers).

Hit. Measurement from a passive sensor or return from an active sensor judged to be from an object, e.g., observation, contact report, return, signal detection, and threshold exceedance.

Hit Assessment. A process that examines the results of an engagement and determines if the target of interest was physically hit. This term has specific meaning for "hit-to-kill" types of engagements where there is no proximity effect and an impact is necessary to damage or destroy the target. A "hit" is not a kill, but is a prerequisite for a kill with hit-to-kill intercepts.

Hit To Kill (HTK). See Kinetic Kill Vehicle

Hostile Track. The classification assigned to a track that, based upon established criteria, is determined to be an enemy threat.

Identification (ID). The process of determining that a tracked object is a friendly, neutral, hostile, or unknown object, or the result of that process.

Impact Point Prediction (IPP). Predicted point of impact on the earth's surface of a reentry vehicle, usually specified in terms of circular error probable. Estimate includes perturbing effects of atmosphere and resultant uncertainties.

Information Pull. Transfer of information product(s) to information user(s) in response to a request by and in a time frame defined by the user or their applications.

Information Push. 1. Transfer of information product(s) to information user(s) in response to profile(s) submitted (typically by the commander's staff) in anticipation of a group of information needs. 2. The process of creating a user profile of information requirements for continuous broadcast to an operating unit or supporting entity.

Infrared (IR). Electromagnetic radiations of wavelength between longest visible red (7,000 Angstroms or 7×10^{-4} millimeter) and about 1 millimeter.

Initial Track. The first track formed for an apparent target.

Intercontinental Ballistic Missile (ICBM). A ballistic missile with a range capability from about 3,000 to 8,000 nautical miles. The term is used only for land-based systems to differentiate them from submarine launched ballistic missiles.

Interoperability. Ability of systems, units, or forces to provide services to or accept services from other systems, units, or forces and to use the services so exchanged to operate effectively together. (2) Conditions achieved among communications-electronics systems or communications-electronics equipment when information or services can be exchanged directly and satisfactorily between them and/or their users.

Intermediate Range Ballistic Missile (IRBM). A ballistic missile having a range capability of 1,500 to 3,000 nautical miles.

Joint Tactical Information Distribution System (JTIDS). Joint Service radio system that provides reliable, secure, jam-resistant, high-capacity integrated communications, navigation, and identification capability through the use of direct-sequence spread-spectrum, frequency-hopping, and error detection and correction techniques. One of two transmission devices currently approved to use Link-16 message standards.

J-Series Family of Tactical Data Links. The family of data links based on common data elements, consisting primarily of the J-series messages and the communications protocols and hardware for Link 16 (TADIL J), Link 22, and VMF, as well as point-to-point, multi-point, and radios/satellite broadcast J-series data link capabilities developed in the future.

Kill Assessment. A process that, based on sensor data, examines in real time the results of an engagement and determines whether the warhead was broken open or not. Based on the outcome the battle manager would decide to or not to fire again at that target. Kill assessment is different than lethality assessment or mission kill. Lethality assessment is a delayed response that may take many minutes or an hour or a day to discern from ground-effects detectors and perhaps on-site visits to the missile wreckage. If the missile or warhead is knocked off course so that it won't land in the defended area the engagement is called a "mission kill". The warhead might not be broken open in a mission kill.

Kinetic Energy Weapon (KEW). Uses kinetic or motion energy to kill an object, e.g., rock, bullet, nonexplosively armed rocket, electromagnetic rail gun.

Kinetic Kill Vehicle (KKV). A weapon using a non-explosive projectile moving at very high speed to destroy a target on impact. The projectile may include homing sensors and on-board rockets to improve its accuracy, or it may follow a preset trajectory (as with a shell launched from a gun).

Laser. An active electron device that converts input power into a very narrow, intense beam of coherent visible or infrared light; the input power excites the atoms of an optical resonator to a higher energy level, and the resonator forces the excited atoms to radiate in phase. Derived from Light Amplification by Stimulated Emission of Radiation and classified from Class I - Class IV according to its potential for causing damage to the eye.

Laser Detection And Ranging (LADAR). Technique analogous to radar that uses laser light rather than radio or microwaves. Light is bounced off target and then detected; return beam provides information on target distance and velocity.

Launch Detection. Initial indication by any one of a variety of sensors that a booster has been launched from some point on the surface of the earth, with initial characterization of the booster type.

Launch On Remote. Interceptor launch approach in which fire control data (measurements or state vectors and error covariance) of sufficient quality are provided by an external system and used by local fire control system to calculate fire control solution and launch interceptor before data can be provided by local sensor. Once launched, this concept assumes local sensor will take over from external sensor, and any in-flight updates to interceptor will be computed based on local sensor information.

Launch Point Determination. With computer methods, uses missile track observation to estimate point on earth's surface from which missile was launched, expressed in terms of circular error probable.

Layered Defense. Sets of weapons that operate at different phases in ballistic missile trajectory; first layer of defense (i.e., boost phase) could pass remaining targets on to succeeding layers

Link. Transmission medium that can be wire, coaxial cable, optical fiber, or free space, as in radio systems. Allows any two subscribers in a network to exchange information

generated by one terminal device and received by another. Uplink and downlink refer to free space transmission from an earth station to a communications satellite and back. Satellite-to-satellite relay is referred to as "crosslink."

LINK-11. See TADIL A

LINK-16 (formerly TADIL-J). NATO designation for the US MIL-STD 6016, Tactical Digital Information Link (TADIL) J message standard and defined in STANAGs 5516 and 5616. U.S. Navy uses NATO designation; its use among all U.S. Joint Services when referring to TADIL J has become more common and recently became policy by JS/J-6 direction. MIL-STD-6016 states that TADIL J and Link-16 are equivalent terms when applied to U.S. systems and platforms' however, Link-16 is preferred. It is a secure, high capacity, jam-resistant, node-less data link that uses the Joint Tactical Information Distribution System (JTIDS) transmission characteristics and the protocols, conventions, and fixed-length message formats defined by the JTIDS Technical Interface Design Plan (TIDP).

Local Track. A track that is initiated and updated by a network participating sensor based on observations of the tracked object(s) by its local sensor(s) only.

Long-Wavelength Infrared (LWIR). Thermal radiation emitted by source in electromagnetic spectrum encompassing infrared wavelengths of 6 to 30 microns.

Low Earth Orbit (LEO). Satellites that are at altitudes between 100 and 400 nautical miles. They have short duration revolutions (about 90 minutes), short visibility envelopes (2.5 to 10 minutes over a tracking station), short life spans, and are most subject to orbital perturbations due to atmospheric drag and earth gravitational anomalies.

LPP. Launch Point Prediction

Medium Wavelength Infrared (MWIR). Thermal radiation emitted by a source in the electromagnetic spectrum encompassing infrared wavelengths of 3 to 6 microns.

Message. A message is information sent and received between an origin and a destination. It has a specified number of bits possibly grouped into words, a definite beginning and a definite end, and obeys certain protocols or rules for the sender and receiver to establish channels and agree on various parameters for unambiguous communication. Messages may be encoded, encrypted, and corresponding decoded and decrypted.

Mid-Course Defense Segment (MDS). The portion of the BMDS that defeats ballistic missiles during the period of flight between boost and atmospheric reentry.

Midcourse Guidance. The guidance applied to a missile between termination of the boost phase and the start of the terminal phase of flight.

Midcourse (MC) Phase. That portion of a ballistic missile's trajectory between the boost phase and the reentry phase when reentry vehicles and penaids travel at ballistic trajectories above the atmosphere. During this phase, a missile releases its warheads and decoys and is no longer a single object, but rather a swarm of RVs and penaids falling freely along present trajectories in space.

Missile Defense Agency (MDA). This agency is tasked by the Secretary of Defense to develop and field the BMDS. The Secretary of Defense directed that the change in names from the Ballistic Missile Defense Organization (BMDO) to MDA in a memorandum dated 02JAN02.

Molniya Orbit. This is a highly eccentric orbit with high apogee (.71 to .74) in the northern hemisphere and low perigee in the southern hemisphere. For a specific set of orbital parameters, this orbit has a changing velocity and altitude, which, when combined with the earth's rotation, keeps the orbiting satellite within view for very long periods (96 percent) above a designated point on earth.

Network. An interlinked web of switching and transmission systems connected to subscriber communications terminals. A network includes all the hardware and software components residing in switching and transmission systems, as well as the communications-related hardware and software and components residing in hosts (e.g., communications protocols).

Network Centric. A term used to describe the widespread sharing of situation information without knowing in advance what value may be derived when that information is available for operational decisions. Information may be shared by a combination of "push" (publish) and "pull" (subscribe) techniques. The shared information is viewed as having no owner, but rather available to all with a need. This sharing allows all war fighters to have the same understanding of the situation and view of the battle space, and facilitates Network Centric Warfare -- integrated operations and synchronization of actions. In Network Centric Warfare the information to be shared is distinguished from command authority authorizing use of the information for combat operations (command and control).

The information shared, together with TTP, the command hierarchy, and commanders' orders provide for Network Centric Operations.

Network Centric Warfare (NCW). An information superiority-enabled concept of operations that generates increased combat power by networking sensors, decision makers, and shooters to achieve shared awareness, increased speed of command, higher tempo of operations, greater lethality, increased survivability, and a degree of self-synchronization. In essence, NCW translates information superiority into combat power by effectively linking knowledgeable entities in the battle space.

Node. A set of equipment and processes, which performs the communications functions at the end of the data links which interconnect those elements, which are resident on the network.

Object. A distinct entity with a definite spatial extent and whose different parts maintain their relative distances constant over a period of observation

Observation Interval. The time that elapses between successive observations of an object by one or more sensors.

PAC-3. PATRIOT Advanced Capability-3

PADIL. Patriot Data & Information Link.

Passive. In surveillance, an adjective applied to actions or equipment, which emit no energy capable of being detected.

Passive Sensor. Detects naturally occurring emissions from target for tracking and/or identification.

Phased-Array Tracking Radar Intercept On Target (PATRIOT). Point or limited area defense system originally built to intercept aircraft. PAC-3 improvements, which will give it greater capability against theater ballistic missiles, include radar upgrades and selection of an improved missile, either PATRIOT multimode or ERINT.

Post-Boost Phase (PBP). That portion of the trajectory of a ballistic missile between the end of powered flight and release of the last RV. Applies only to multiple-warhead ballistic missiles.

Post-Boost Vehicle (PBV). The portion of a rocket payload that carries multiple warheads and which has the maneuvering capability to independently target each warhead on a final trajectory toward a target. Also referred to as a "bus."

Precision Decoys. Decoys that precisely match RV characteristics either exo-atmospherically or endo-atmospherically, or both, and seek to deceive the defense into intercepting them.

Predicted Intercept Point (PIP). The calculated position in space where the target and interceptor coincide.

Probability of Detection (P_d). The probability that an observation is generated from a frame of sensor data for an object that is within the field of view.

Probability of Kill (P_k). Describes the lethality of a weapon system. Generally refers to armaments (i.e. missiles, ordnance, etc.) Usually the statistical probabilities that the weapon will detonate close enough to the target with enough power to disable the target

Protocol. Rules, such as open systems interconnection, that enable error-free computer connection and communication at a given layer or segment of a network architecture. Typically established by industry or international organizations such as the Institute of Electrical and Electronic Engineers (IEEE) or American National Standards Institute (ANSI).

Radar. (Formerly an acronym for Radio Detection and Ranging.) A technique for detecting targets in the atmosphere or in space by transmitting radio waves (e.g., microwaves) and sensing the waves reflected by objects. The reflected waves (called "returns" or "echoes") provide information on the distance to the target and the velocity of the target, and also may provide information about the shape of the target.

Real Time. Pertaining to the timeliness of data or information that has been delayed only by the time required for electronic communication. This implies that there are no noticeable delays.

Reentry. The return of objects originally launched from earth, into the atmosphere.

Reentry Phase. That portion of the trajectory of a ballistic missile or space vehicle where there is a significant interaction of the vehicle and the earth's atmosphere.

Reentry Vehicle (RV). 1. A structure designed to return from exo-atmospheric flight through Earth's atmosphere. 2. Reentry vehicles are objects containing nuclear, chemical, biological, or high explosive warheads. They are released from the last stage of a booster rocket or from a post-boost vehicle early in the ballistic trajectory. They are

likely thermally insulated to survive rapid heating during reentry into the atmosphere.

Remote Track. A track that consists of data only from one or more non-organic sensors.

Robust. Used in describing a system; indicates its ability to endure and perform its mission against a responsive threat. Also used to indicate system ability to survive under direct attack.

Robustness. 1. The ability to produce correct results despite input errors. 2. The existence of coordinated, multiple capabilities that perform the same broad task/mission. Provides the BMD warfighter with sufficient flexibility to negate the specified threat with application of a variable mix of ground and space-based systems.

Rules Of Engagement (ROE). Directives issued by competent military authority which delineate the circumstances and limitations under which United States forces will initiate and/or continue combat engagement with other forces encountered.

SBIRS High. SBIRS high altitude component consisting of four SBIRS GEO satellites and infrared sensors on two HEO satellites.

SBIRS Low. SBIRS low altitude component consisting of SBIRS LEO satellites. The SBIRS Low component will be designed to provide precision midcourse tracking and discrimination data to support early interceptor commit, in-flight target updates, and target object maps for a National Missile Defense architecture. The SBIRS Low component will also support the other mission areas of the SBIR system.

Sensor. A device that responds to a physical stimulus (as heat, light, sound, pressure, magnetism, or a particular motion) and transmits a resulting impulse (as for measurement or operating a control).

Sensor Data. Measurement information. For a passive sensor it is usually irradiance, time, azimuth angle and elevation angle. For an active sensor it may include range, Doppler, cross section, etc., as well.

Sensor Fusion. Combining data and information from multiple sensors, usually on different platforms. Processing for doing this for two passive sensors is sometimes called stereo fusion and for three passive sensors, triocular or triple fusion. The term may also be applied to passive-active or active-active fusion as well.

Sensor Network. 1. All external and internal ballistic missile defense system sensors plus comms and sensor netting used to communicate sensor data to algorithms, processes, and people who use it. 2. All ballistic missile defense system sensors, internal and external to Missile Defense Agency.

Sensor Node. Sensor-netting node collocated with sensor that provides target track and feature information to sensor netting network and receives sensor tasking information from it.

Shoot-Look-Shoot (SLS). A firing doctrine in which the result of the first intercept attempt is assessed prior to the launch of a subsequent interceptor. This tactic requires the use of kill assessment by space or ground based sensors but can significantly reduce interceptor inventory requirements.

Short-Range Ballistic Missile (SRBM). A ballistic missile with a range capability of 30 km to 1,000 km.

Short Wavelength Infrared (SWIR). Thermal radiation emitted by a source in the electromagnetic spectrum encompassing infrared wavelengths of 0.75 to 3 microns.

Shorting. With regard to the Sensor Net, it is the action of sending tracks to the weapons platform at the same time that they are sent through the TCC/TRC/TSC processing loop.

Single Integrated Air Picture (SIAP). 1. The SIAP is the product of sensor fused, common, continual, unambiguous tracks of airborne objects in the surveillance area. Each object in the SIAP has one, and only one, track number and set of associated characteristics. The SIAP uses fused, near real-time and real-time data, scalable and filterable, to support situational awareness, battle management, and airborne target engagements. 2. The SIAP (the air track portion of the CTP) consists of common, continual, and unambiguous tracks of airborne objects of interest in the surveillance area. SIAP is derived from real time and near real time data and consists of correlated air object tracks and associated information. The SIAP uses fused near real time and real time data, scaleable and filterable, to support situation awareness, battle management, and target engagements. 3. The SIAP is the product of fused, common, continuous, unambiguous tracks of all airborne objects in the surveillance area. Each object within the SIAP has one, and only one, track number and set of associated characteristics. The SIAP is developed from near-real-time and real-time data, and is scaleable and filterable to

support situation awareness, battle management, and target engagements. 4. As in 3) above plus ... The SIAP is a subset of the CTP, used by TAMDC² and weapon control nodes to share track and fire-control data.

Space Based Infrared System (SBIRS). SBIRS will be a consolidated system that will meet United States infrared space surveillance needs through the next 2-3 decades. SBIRS is intended to be an integrated "system of systems" including multiple space constellations and an evolving ground element. The baseline SBIRS architecture consists of four Geosynchronous Earth Orbit (GEO) satellites; two sensors on Highly Elliptical Orbit (HEO) satellites; Low Earth Orbit (LEO) satellites; a ground system consisting of a CONUS-based Mission Control Station (MCS), a backup MCS, a survivable MCS, and oversees relay ground stations and re-locatable terminals; and associated communications links. The SBIRS is designed to meet the missile defense, missile warning technical intelligence, and battle space characterization mission requirements identified in the JROC-validated SBIRS Operational Requirements Document. The SBIRS program will begin replacing the operational Defense Support Program (DSP) ground segment in 1999 and begin replacing the DSP satellites in 2002.

Space-Based Sensor. A system that provides global above-the-horizon surveillance to detect and track PBVs, object clusters (RVs and penaids), and resolved midcourse objects, as well as below-the-horizon tasked hot spot detection of boost phase missiles when cued by a space-based weapon or a *priori* knowledge. It provides surveillance data for use in situation assessment, operational intelligence collection, and for cueing other sensor and weapon elements. During midcourse, sensors discriminate and track RVs and associated objects to support midcourse engagements.

Standard Missile. A shipboard, surface-to-surface/air missile.

Surveillance. The systematic observation of an aerospace area by a sensor system primarily for the purpose of detecting an air vehicle, ballistic missile, or other aerospace object. Some sensors that are referred to as surveillance sensors also track air vehicles, ballistic missiles, and other aerospace objects.

System. 1. The organization of hardware, software, materials, facilities, personnel, data, and services needed to perform a designated function with specified results, such as the gathering of specified data, its processing, and delivery to users. 2. A combination of two or more interrelated equipment (sets) arranged in a functional

package to perform an operational function or to satisfy a requirement.

System Architecture. The structure and relationship among the components of a system. The system architecture may also include the system's interface with its operational environment. A framework or structure that portrays relationships among all the elements of missile defense systems.

System of Systems (SoS). A set or arrangement of interdependent systems designed to be interconnected in various ways to provide capabilities beyond those systems operating autonomously. Each component system is designed with a "fall back" capability to operate autonomously, but when operated as an interconnected set, their capabilities are enhanced. The degree of interdependence can vary from loosely coupled (federated) to tightly coupled (integrated), but the capability of the set is always greater than the sum of the elements operated autonomously.

Tactical Data Links. Near-real-time tactical communications and information systems used primarily at the coordination and execution level.

Tactical Digital Information Link (TADIL). A Joint Staff approved, standardized communication link suitable for transmission of digital information. Current practice is to characterize a tactical digital information link (TADIL) by its standardized message formats and transmission characteristics. TADILs interface two or more command and control or weapons systems via a single or multiple network architecture and multiple communication media for exchange of tactical information.

TADIL A. A secure, half-duplex, netted digital data link utilizing parallel transmission frame characteristics and standard message formats at either 1364 or 2250 bits per second. It is normally operated in a roll-call mode under control of a net control station to exchange digital information among airborne, land-based, and shipboard systems. NATO's equivalent is Link 11

Target. 1. Same as defined for an object. 2. An object of interest rather than just any type of object.

TBM. Tactical/Threat Ballistic Missile

TBMD. Tactical/Theater Ballistic Missile Defense

Theater Ballistic Missile Defense (TBMD) System. The aggregate TMD C3I and TBMD forces that, in total, provide defense against ballistic missile attacks within an overseas theater of operations.

THAAD. Theater High Altitude Area Defense

Terminal Phase. That final portion of a ballistic missile's trajectory between the midcourse phase and trajectory termination.

Track. 1. The estimated position/velocity states and a representation of the uncertainty of the estimate (and possibly additional non-kinematic attribute information) for an object or unresolved cluster of objects based on filtered observations from one or more sensors. 2. The estimated trajectory of an apparent object or group of objects. 3. The sequence of observations judged to be from the same object or group of objects.

Track Correlation. Process of associating multiple tracks from each of two different sensors and determining track pairs that represent the same objects. Track data (position, velocity, signature attributes, etc.) from one sensor are compared with those from the second sensor. Prevents/eliminates dual designations. Correlated track pairs can be combined to refine target position/velocity estimates.

Track File. A dataset that contains data associated with a target track, including metric measurements, signature data, state estimate, covariance matrix, track quality, class/type, and other attributes of the target.

Track Fusion. Merging of 2D tracks or 3D tracks from different sensors to form more precise 3D tracks.

Track Quality. A quantitative or qualitative measure of the reliability or credibility of a track.

Tracking. Following a target in angle, range, and Doppler. Usually involves measuring target position, smoothing position measurements to obtain a more accurate assessment of target position, predicting target position ahead in time, and using that prediction to gather next sample measurement.

Track Initiation. The process of inferring new target trajectories. It typically consists of the association of several detections over time and a decision that accepts these detections as having originated from the same target.

Track Update. The combination of a track and an observation to form a revised track.

Triangulation. The process by which the range to a target is inferred from observations from two or more sensors. Compare: passive ranging.

Universal Time. A measure of time that conforms, within a close approximation, to the mean diurnal rotation of the Earth and serves as the basis of civil timekeeping. Universal Time (UT1) is determined from observations of the stars, radio sources, and also from ranging observations of the moon and artificial Earth satellites. The scale determined directly from such observations is designated Universal Time Observed (UTO); it is slightly dependent on the place of observation. When UTO is corrected for the shift in longitude of the observing station caused by polar motion, the time scale UT1 is obtained. When an accuracy better than one second is not required, Universal Time can be used to mean Coordinated Universal Time. Also called ZULU time. Formerly called Greenwich Mean Time.

Weapons Allocation. Designation of a certain weapon to attack a certain threat after Engagement Authorization is given.

Weapons Assignment. In air defense, the process by which weapons are assigned to individual air weapons controllers for use in accomplishing an assigned mission. Assignment of a particular interceptor to a particular target.

Weapons Control. The varying degree of formal control an area air defense commander exercises over all air defense weapons in his area of responsibility.

Weapons of Mass Destruction (WMD). In arms control usage, weapons that are capable of a high order of destruction and/or of being used in such a manner as to destroy large numbers of people.

Weapons System. Items that can be used directly by the armed forces to carry out combat missions and that cost more than \$100,000 or for which the eventual total procurement cost is more than \$10,000,000. That term does not include commercial items sold in substantial quantities to the general public.

Weapon System Control. That set of assessment, decision, and direction functions normally implemented automatically to assure that individual weapons are pointed, fired, and guided as necessary to intercept the designated attackers.

**Ballistic Missile
Defense System
(BMDS)**

**Vision Document
Version 1.0**

Revision History

Date	Revision	Description	Author
03/27/2003	1.0	Initial Version	Babbitt
12/01/2003	1.0	Thesis Update	Miklaski

1. Introduction.

1.1 Purpose of Document.

This document outlines the high-level user requirements and features of the Ballistic Missile Defense System (BMDS).

1.2 Product Overview.

BMDS will enable the United States, its allies, and friends to detect, track, assign weapons to, engage, and assess the kill of threat ballistic missiles in the boost, mid-course and terminal phases of missile flight in a rapid, coordinated, and effective manner.

1.3 References.

Refer to Thesis Appendix B.

2. Problem Statement

Problem	
The problem of	Current BMD efforts are uncoordinated and lack the ability to respond quickly to potential threats
affects	The U.S.'s confidence in its ability to defend against missiles and, ultimately, the safety, security, and prosperity of the free world
The impact of which is	Allows rogue states and entities to blackmail or attack free nations
a successful solution would be	To field a system that will counter the ballistic missile threat with a high level of confidence

3. User Description.

3.1 User Demographics.

The BMDS Battle Managers do not have a tool that supports ballistic missile defense operations for detecting, tracking, assigning weapons, engaging, and assessing the kill of threat ballistics missiles from potential adversaries.

3.1.1 Northern Command Battle Managers.

3.1.2 Strategic Command Battle Managers.

3.1.3 Combatant Commanders Battle Managers.

3.1.4 Assigned Forces Battle Managers.

3.2 User Profiles.

3.3 User Environment.

3.4 Key User Needs.

The BMDS Battle Managers require the following capabilities:

3.4.1 Detect the launch of a threat ballistic missile.

3.4.2 Determine whether the detected object is a threat.

3.4.3 Define the characteristics of the threat ballistic missile.

3.4.4 Develop a firing solution to negate the threat ballistic missile.

3.4.5 Engage the threat ballistic missile.

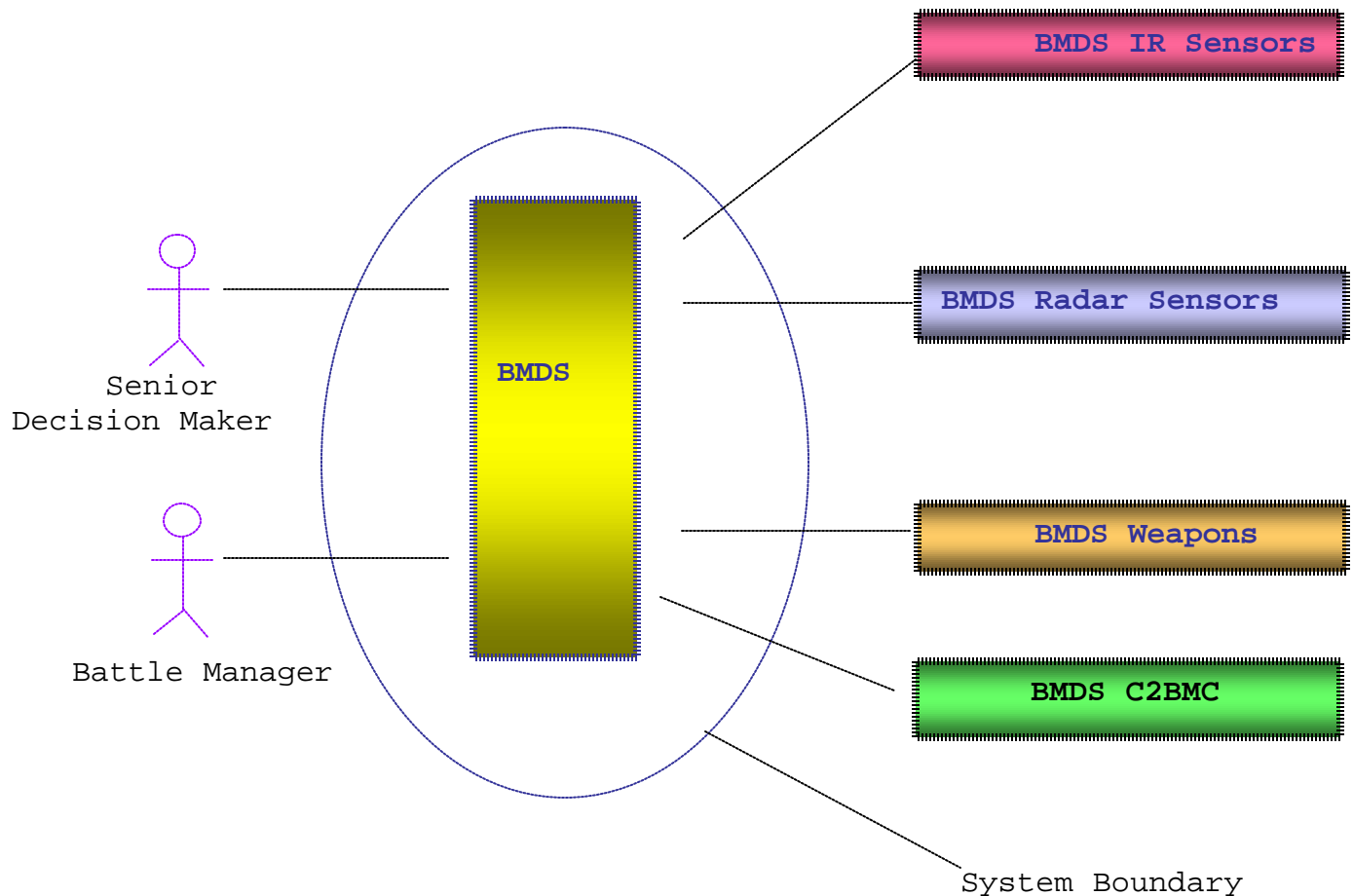
3.4.6 Assess the kill of the threat ballistic missile.

3.5 Alternatives.

Without such a tool, current battle management functions will remain autonomous functions that are independent of each other.

4. Product Overview.

Product Perspective. The below diagram depicts the BMDS virtual simulation and its external interfaces.



The diagram on following page represents the functional aspects of the BMDS Kill Chain. The Battle Manager must address the battle management functions identified on the BMDS Kill Chain.

Refer to Thesis Chapter 4 for Kill Chain Description.

5. Use Cases.

Refer to Thesis Chapter 4 for description of BMDS Use Cases.

6. Feature Attributes.

The table below contains the feature attributes that we will use to evaluate the features, track the continued feature definition, prioritize the risk, and manage the feature requirements.

Feature Attribute	Scale
Status	Designed, Approved, Proposed
Realization	Full, Partial, Limited
Priority	Critical, Important, Useful
Complexity	High, Medium, Low
Risk: Probability of Occurrence	High, Medium, Low
Risk: Consequence of Occurrence	Catastrophic, Significant, Minor
Stability	High, Medium, Low

7. Product Features.

7.1 Forward-based sensing.

Within BMDS we must design the capability to employ forward-based sensors to pickup the IR detection of a threat ballistic missile. Without this capability, the BMDS Battle Managers will not have the ability for tracking the threat ballistic missile from booster burnout through the mid-course and terminal phases of missile flight. This situation would result in track discrimination and a first shot opportunity either late in the mid-course phase or in the terminal phase.

7.2 Track correlation.

Given that two or more radars may provide real-time track data for a single threat ballistic missile, BMDS must be able to match this track data so that each threat ballistic missile in flight results in a single, accurate reported track in the battle management system.

7.3 Common Time Reference.

All sensors employed by BMDS must have a common time reference to provide accurate track data for track correlation algorithms.

7.4 Common Navigation Reference.

All sensors employed by BMDS must employ a common geodetic navigation scheme to ensure accurate position, velocity, and altitude adjustments in real-time threat ballistic missile tracking.

7.5 Sensor Registration.

All sensors employed by BMDS must have a common alignment reference to ensure the correct geodetic alignment of the sensors for the objective of establishing gridlock.

7.6 Cueing.

BMDS must have the capability to direct sensors it employs to adjust radar field-of-views towards IR-detected track.

7.7 Discrimination.

BMDS must have the capability to accurately discriminate in real-time the threat ballistic missile from other objects such as deployed countermeasures and debris. BMDS must provide the discrimination processes through all phases of threat ballistic missile flight.

7.8 Multi-Sensor Data Fusion.

BMDS must have the capability to fuse data from multiple, autonomous sensors with the intent of forming a more accurate estimation of the environment than is available from any single sensor. The data fusion capability is time critical, covers a large geographical area, and requires accurate, reliable information at completion.

7.9 Information Assurance.

Information within BMDS must be secure both in transfer and processing to ensure data integrity throughout the kill chain. Network and information services must ensure confidentiality and availability.

7.10 Assurance of Kill.

BMDS must provide assurance of kill within statistically acceptable limits.

7.11 Speed of Engagement.

BMDS must provide the ability to simultaneously respond to multiple missile launches and assigned available weapons faster than a human controller executing the same process. This must be done within the framework of system weapons' engagement windows.

7.12 Automatic Weapons Assignment.

BMDS must have the capability to automatically match its weapon capabilities and threat missile profiles in such a fashion that assets are properly assigned to ensure the maximum chance of destroying the threat missile or missiles.

7.13 Situational Awareness.

BMDS must provide real time, fine grain situational awareness to battle managers responsible for the kill chain in each portion of a threat missile's flight as well as near real time, accurate rough grain situational awareness to all battle managers not in the geographic path of flight or without responsibility for the managing of the kill chain pertinent to a particular missile. This must be done on a threat-by-threat basis.

7.14 Fault Tolerance.

BMDS must be capable of continuing operation in a degraded mode following a catastrophic failure or loss of any of its individual elements in a dynamic fashion, providing the remaining elements the ability to continue without the lost element(s).

7.15 Cooperative and Autonomous Modes.

The structure of BMDS must support autonomous (independent) action at the lowest level where the entire kill chain can be executed. It must also support low cooperative, high cooperative, and fully cooperative modes of command and control.

7.16 Combined Operation with Coalition Forces.

BMDS must provide a means for the U.S. to coordinate its efforts with the forces of our allies and friends.

7.17 Dynamic Reconfigurability.

As BMDS enabled and integrated assets move from one entity's control to another entity's control, they must seamlessly integrate into the new control structure.

8.Constraints.

To be determined.

9. Performance Requirements.

To be determined.

10. Dependencies.

To be determined.

11. Documentation Requirements.

To be determined.

12. Issues.

To be determined.

13.Glossary.

Refer to Thesis Appendix A

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. SRS DOCUMENT

Ballistic Missile
Defense System

Software Requirements Specification
(SRS)
Version 1.0

Revision History

Date	Revision	Description	Author
03/27/2003	1.0	Initial Version	Miklaski
12/01/2003	1.0	Thesis Update	Miklaski

Table of Contents

- 1.0 Introduction
 - 1.1 Purpose
 - 1.2 Scope
 - 1.3 References
 - 1.4 Assumptions and Dependencies
- 2.0 Use Case Model Survey
- 3.0 Actor Survey
- 4.0 Requirements
 - 4.1 Functional Requirements
 - 4.1.1 Sensors
 - 4.1.2 Weapons
 - 4.1.3 Command and Control
 - 4.2 Nonfunctional Requirements
 - 4.2.1 Usability
 - 4.2.2 Reliability
 - 4.2.3 Performance
 - 4.2.4 Supportability
- 5.0 User Documentation
- 6.0 Design Constraints
- 7.0 Interface Components
- 8.0 Interfaces
 - 8.1 User Interfaces
 - 8.2 Hardware Interfaces
 - 8.3 Software Interfaces
 - 8.4 Communications Interfaces
- 9.0 Appendix
 - 9.1 Use Case Diagrams
 - 9.2 System Sequence Diagrams
 - 9.3 Domain Model
 - 9.4 Glossary

1.0 Introduction.

1.1 Purpose.

The purpose for this Software Requirements Specification (SRS) is to define the requirements for the Ballistic Missile Defense System (BMDS). The SRS includes Use Cases, System Sequence Diagrams, System Operations Contracts, Domain Model and all related support documentation to describe the functionality of the BMDS. The design team of CDR Michael Miklaski and CPT Joel Babbitt has prepared the SRS and its related documentation.

1.2 Scope.

The BMDS is being developed via a system-of-systems approach to better integrate sensors, weapons and command and control nodes into a single coherent command and control structure. BMDS requires an advanced and highly complex command and control element to effectively integrate system segments and execute battle management functions. The BMDS architecture is designed to accept enhanced capabilities as they are integrated into the BMDS, to achieve full interoperability of the system elements and interfaces with external systems and integrates the system with the national military command structure.

1.3 References.

See Thesis Appendix B.

1.4 Assumptions and Dependencies.

1.4.1 Assumptions.

1.4.1.1 Non-instantiation. No specific system is used to define the class. All sensors, weapons, and C3 structures are generic in nature and have only those attributes that define the common functionality.

1.4.2 Dependencies.

1.4.2.1 Kill Chain. The flow of events is dictated by the logical sequence represented in the Kill Chain Diagram (see 9.3.1)

2.0 Use Case Model Survey.

See Thesis Chapter 4.

3.0 Actor Survey.

Sensors - Within the sensor actors' category there are two subcategories. The first is Passive Infrared Sensors; the second is Active Radar Sensors. Within each of these subcategories are the platform on which they reside; Space Based, Aircraft Based and Surface Based.

The current state of Space Based Sensors are satellite buses with Infrared (IR) sensor payloads that can detect and track the heat from the plume of a ballistic missile during the launch and boost phase. There are currently only passive IR sensors in Space, which can only provide lines of bearing and altitude. Precise ranging is not feasible unless some form of triangulation occurs with other Sensors. Research is being conducted to determine if the specific heat signature of the ballistic missile can identify the actual type of missile launched. Additional research is being conducted to determine the feasibility of "cold body" tracking of ballistic missile. This is when the missile is in the cruise phase and the IR signature is reduced through the cooling of the skin by the cold temperatures of space. The primary mission of Space Based Sensors is to provide initial cueing information to active sensors and to determine, based on launch position and angular motion, the intended target location.

Airborne Sensors consist of both passive IR and active radar systems. The developmental Airborne Laser System functions in much the same way as the Space Based Sensor with the exception of being able to determine range based on its Laser Detection and Ranging (LADAR) system. Other Airborne Based Sensors consist of active radars and passive IR sensors onboard aircraft.

Surface Based Sensors consist of existing active radars associated with Patriot, THAAD, AEGIS, National Systems, and developmental X-Band radar that can track ballistic missiles during the different stages of flight. This category encompasses both ground and sea based sensor platforms.

Weapons - Three separate subsets, ballistic, semi-active homing and active homing define the weapons actors. The weapons may have an explosive payload, be a Directed Energy weapon, or a Kinetic Kill Vehicle (KKV). Ballistic weapons

require the weapons system to develop a collision intercept solution prior to deploying the weapon, then firing the weapon along a trajectory that will consummate an impact with the threat ballistic missile. Semi-active weapons require that the weapons system utilize an active sensor, normally Continuous Wave (CW) Radar, to illuminate the threat missile and the weapon guides on the return signal until end game. Active weapons receive queuing data from the weapons system prior to launch and will initially track the target in the same way as the semi-active weapon until it can acquire the target with its own active radar. Once it acquires the target with its own radar it will discontinue semi-active and guide to the target utilizing its own information.

Command and Control - These actors encompass all existing command and control nodes from the National Level, such as STRATCOM and SPACECOM, to the tactical commander level, such as Joint Forces Air Component Commander (JFACC), Carrier Battle Group Commanders (CVBG), Regional Air Defense Commanders (RADC), etc. The level of information provided to the commanders is based on hierarchy. The lower levels of command will require the most current and accurate information to prosecute the destruction of Threat Ballistic Missiles (TBM's) and will need to receive time-critical and precise parametric tracking data to achieve that goal. Upper levels generally require only situational awareness information and will not need precise information such as parametric data and instead will be provided those necessary data to ensure proper execution for command and control functionality. The need to know either high level or lower level information is determined by if one is involved in the kill chain for a missile and where the missile is in the phases (boost, midcourse, terminal).

Threat Ballistic Missiles - For the purposes of our requirements specifications the category of ballistic missile will include only those missiles that travel through the exo-atmospheric region of space. This excludes short-range tactical missiles that remain in the endo-atmospheric region.

4.0 Requirements

4.1 Functional Requirements

4.1.1 Sensors

4.1.1.1 IR

- 4.1.1.1.1 Determine own position and global time accurately
- 4.1.1.1.2 Detect the plume of a launching ballistic missile
- 4.1.1.1.3 Track a ballistic missile from heat signature
- 4.1.1.1.4 Identify type of missile from heat signature
- 4.1.1.1.5 Accept queuing from external sources
- 4.1.1.1.6 Provide queuing information to external sensors
- 4.1.1.1.7 Develop a ballistic missile track
- 4.1.1.1.8 Continuously track missile through field of view
- 4.1.1.1.9 Determine launch position of detected Ballistic Missile
- 4.1.1.1.10 Determine Predicted Impact Point (IPP)
- 4.1.1.1.11 Transmit all known track data and own unit position to external units.

4.1.1.2 Radar

- 4.1.1.2.1 Determine own position and global time accurately
- 4.1.1.2.2 Accept queuing information from external sources
- 4.1.1.2.3 Detect ballistic missile in flight
- 4.1.1.2.4 Track ballistic missile in flight
- 4.1.1.2.5 Develop a ballistic missile track
- 4.1.1.2.6 Provide track information to external sources
- 4.1.1.2.7 Provide weapons quality parametric data to weapons system
- 4.1.1.2.8 Continuously track missile through field of view
- 4.1.1.2.9 Provide queuing information to external sensors
- 4.1.1.2.10 Transmit all known track data and own unit position to external units.
- 4.1.1.2.11 Assess kill

4.1.2 Weapons

4.1.2.1 Ballistic

- 4.1.2.1.1 Determine own position and global time accurately
- 4.1.2.1.2 Accept queuing information from external sources
- 4.1.2.1.3 Accept target tracking parametric information from sensors
- 4.1.2.1.4 Develop a collision intercept solution
- 4.1.2.1.5 Slew the weapon to corresponded to the intercept solution
- 4.1.2.1.6 Fire projectile

4.1.2.2 Semi Active

- 4.1.2.2.1 Determine own position and global time accurately
- 4.1.2.2.2 Accept queuing information from external sources
- 4.1.1.2.3 Accept target tracking parametric radar information from sensors

- 4.1.1.2.4 Develop a collision intercept solution
- 4.1.1.2.5 Slew weapon seeker head to target
- 4.1.1.2.5 Fire missile
- 4.1.1.2.5 Home to target with return radar signal from sensors

4.1.2.3 Active

- 4.1.1.3.1 Determine own position and global time accurately
- 4.1.1.3.2 Accept queuing information from external sources
- 4.1.1.3.3 Accept target parametric radar data from sensors
- 4.1.1.3.4 Develop a collision intercept solution
- 4.1.1.3.5 Slew weapon seeker head to target
- 4.1.1.3.3 Fire weapon
- 4.1.3.3.3 Track in a semi-active mode based on return radar signal from sensors
- 4.1.3.3.4 Acquire target with missile own radar
- 4.1.3.3.5 Discontinue semi-active tracking upon own radar lock and assume collision intercept based on missile own radar parametric data

4.1.3 Command and Control

4.1.3.1 Upper Level.

- 4.1.3.1.1 Accept Situational Awareness tracking data from all external sources to develop a single Common Operational Picture
- 4.1.3.1.2 Transmit commands to Lower Level C2 nodes
- 4.1.3.1.3 Receive reply messages from Lower Level C2 Nodes

4.1.3.2 Lower Level.

- 4.1.3.2.1 Accept tracking data from all external sources to develop a single Common Operational Picture
- 4.1.3.2.2 Fuse all sensor data into a single actionable firing solution
- 4.1.3.2.3 Transmit all Situational Awareness tracking data to Upper Level C2 Nodes
- 4.1.3.2.4 Accept command messages from Upper Level C2 Nodes
- 4.1.3.2.5 Transmit reply messages to Upper Level C2 Nodes

4.2 Nonfunctional Requirements.

4.2.1 Usability.

To be determined.

4.2.2 Reliability.

To be determined.

4.2.3 Performance.

To be determined.

4.2.4 Supportability.

To be determined.

5.0 User Documentation.

To be determined.

6.0 Design Constraints.

To be determined.

7.0 Interface Components.

To be determined.

8.0 Interfaces.

8.1 User Interfaces.

To be determined.

8.2 Hardware Interfaces.

To be determined.

8.3 Software Interfaces.

To be determined.

8.4 Communications Interfaces.

To be determined.

9.0 Appendix.

9.1 Use Case Diagrams.

See Thesis Chapter 4.

9.2 System Sequence Diagrams.

See Thesis Appendix E.

9.3 Domain Models.

9.3.1 BMDS Kill Chain Functions.

See Thesis Figure 1.

9.3.2 BMDS Distributed C2 Architecture.

See Thesis Figure 5.

9.4 System Operations Contracts.

To be determined.

9.5 Glossary.

See Thesis Appendix A for Glossary.

APPENDIX D. SYSTEM SEQUENCE DIAGRAMS (SSD)

A. SSD FOR HIGH-LEVEL BMDS USE CASE

This SSD (Figure 14) represents the high-level interaction of the sensors, weapons, and BMC2 through the five major phases of the kill chain in the prosecution a threat ballistic missile to include deliberate planning and cueing. The details of this SSD are further defined in the follow-on SSD for all the use cases. The flow of events as described in the use cases and messaging between the objects is as depicted, a BMC2 will provide planning and cueing to sensors in response to a potential threat. The sensors conduct surveillance until a TBM is detected where it then conducts tracking, passing that information both to the BMC2 and weapons systems. The BMC2 assigns a weapon based on the track data, the selected weapon system engages the threat ballistic missile destroying it, and the BMC2 assesses the outcome of the engagement via the sensor track data to determine whether further engagements are warranted.

B. SSD FOR USE CASE 1 & 1.1

The flow of events for the SSD covering use cases 1 & 1.1 (Figure 15) begins with the assumption that a ballistic missile threat exists and that there is a sufficient amount of time to conduct deliberate planning prior to the anticipated first available launch window. In this instance the commanders, via the BMC2 and Sensor Net, issue a warning in the form of cueing messages for sensors to observe a specific region. Once a TBM is detected, the sensor commences continuous tracking of the missile and forwards a cueing message to the BMC2 and Sensor Net so that other sensors can detect and track the TBM.

The sensor must first develop a local track, through whatever processing method that particular sensor utilizes, and then it forwards that data to its associated Sensor Fusion Processor. The SFP receives all of the track data from the various sensors and attempts to both discriminate what type of missile it is and whether any countermeasures have been employed, to detect the warhead from the decoys. The data is further filtered and then fused into one coherent track and forwarded to the Sensor Net for utilization by C2 elements and Weapons systems.

C. SSD FOR USE CASE 2

Once a sensor has developed a track, it is cooperatively tracked and classified as described by the flow of events in the SSD for use case no. 2 (Figure 16). The Sensor Fusion Processor continues to evaluate the track in an effort to determine the identity of the missile through the various electronic signatures from the sensors and attempts to refine and improve the track quality by pulling available data from the Sensor Net and making comparisons of the data. This updated data is forwarded to the BMC2. The BMC2 maintains a master track list is developed and pushes the track data to the Weapons Net for use in the weapons bidding process. The BMC2 forwards an appropriate cueing message to remote Sensor Nets as the TBM transits from sensor coverage area to another.

D. SSD FOR USE CASE 3

The SSD for use case 3 (Figure 17) shows the process for weapons bidding of the weapons systems. As the target list is produced by BMC2 and pushed out to the Weapons Net, the Weapons Net places bid requests for each target to each of the Weapons Systems participating in the network. The Weapons System makes an assessment of it's own ability to prosecute the target and forwards that information to BMC2

via the Weapons Net. The BMC2 then makes a weapons assignment based on the bids and an authorization to release weapons at the appropriate time.

E. SSD FOR USE CASE 4

Once a Weapons System has been identified for missile engagement, that systems requests a discrete priority path for parametric track data from the Sensor Net as depicted in SSD for use case no. 4 (Figure 18). This data is then provided to the interceptor for consummation of the intercept and destruction of a TBM once launch approval is given.

F. SSD FOR USE CASE 5

SSD for use case 5 (Figure 18) depicts the flow of events in the process of assessing the status of an intercept of a TBM. The BMC2 utilizes track data provided via the Sensor Net and conducts an assessment of that data as described in use case no. 5 and promulgates a kill report for distribution if the intercept was successful.

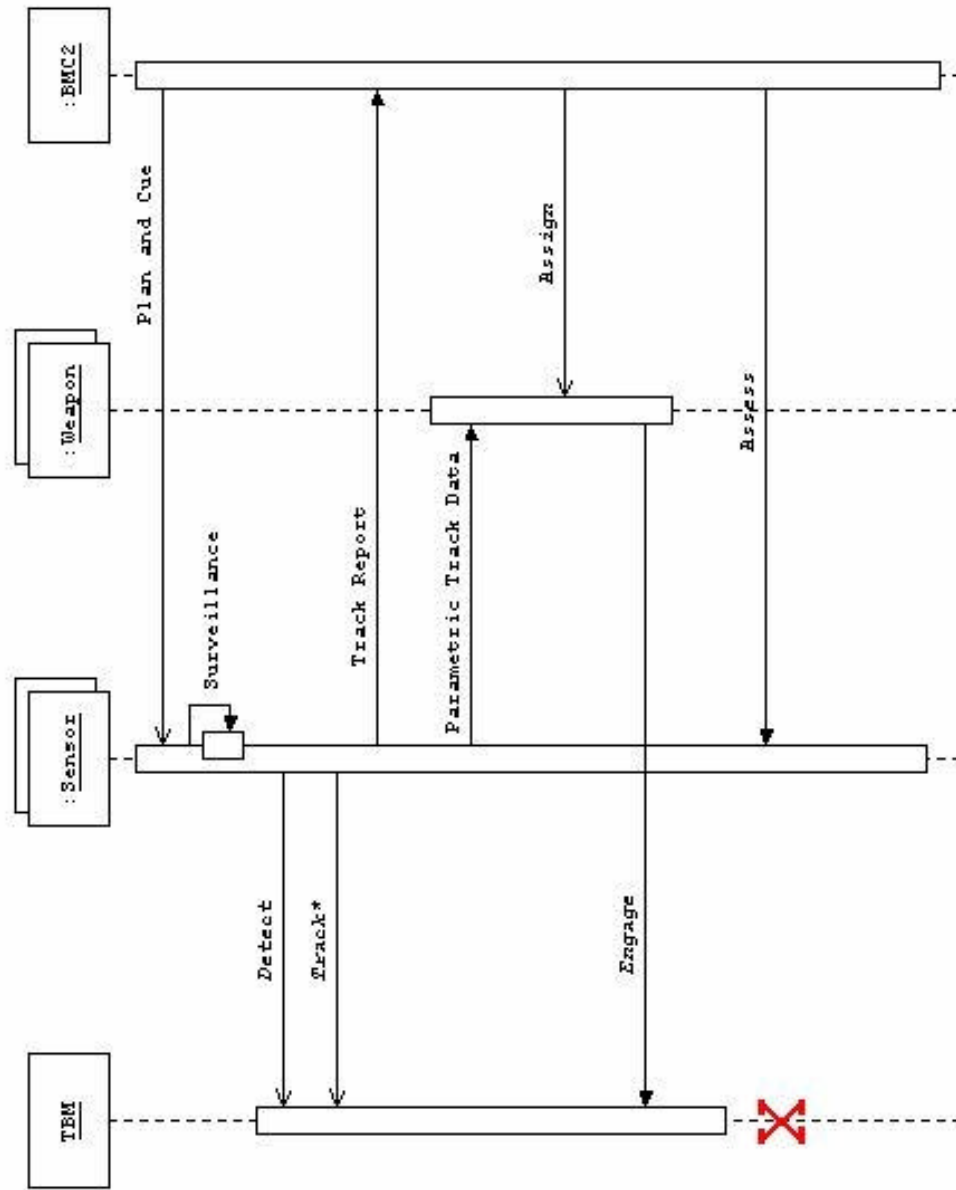


Figure 17. SSD for High-Level Use Case

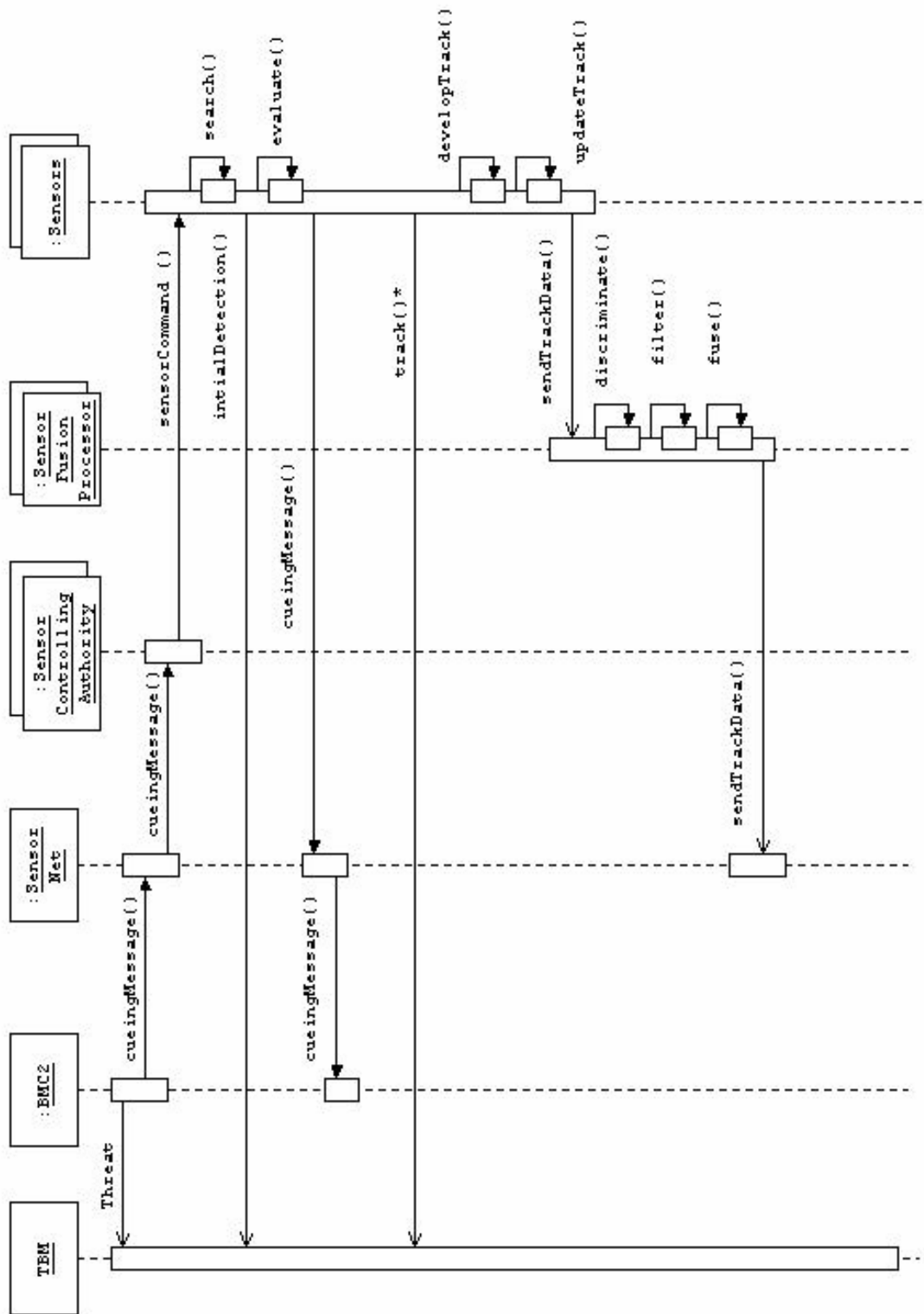


Figure 18. SSD for Use Case 1 & 1.1

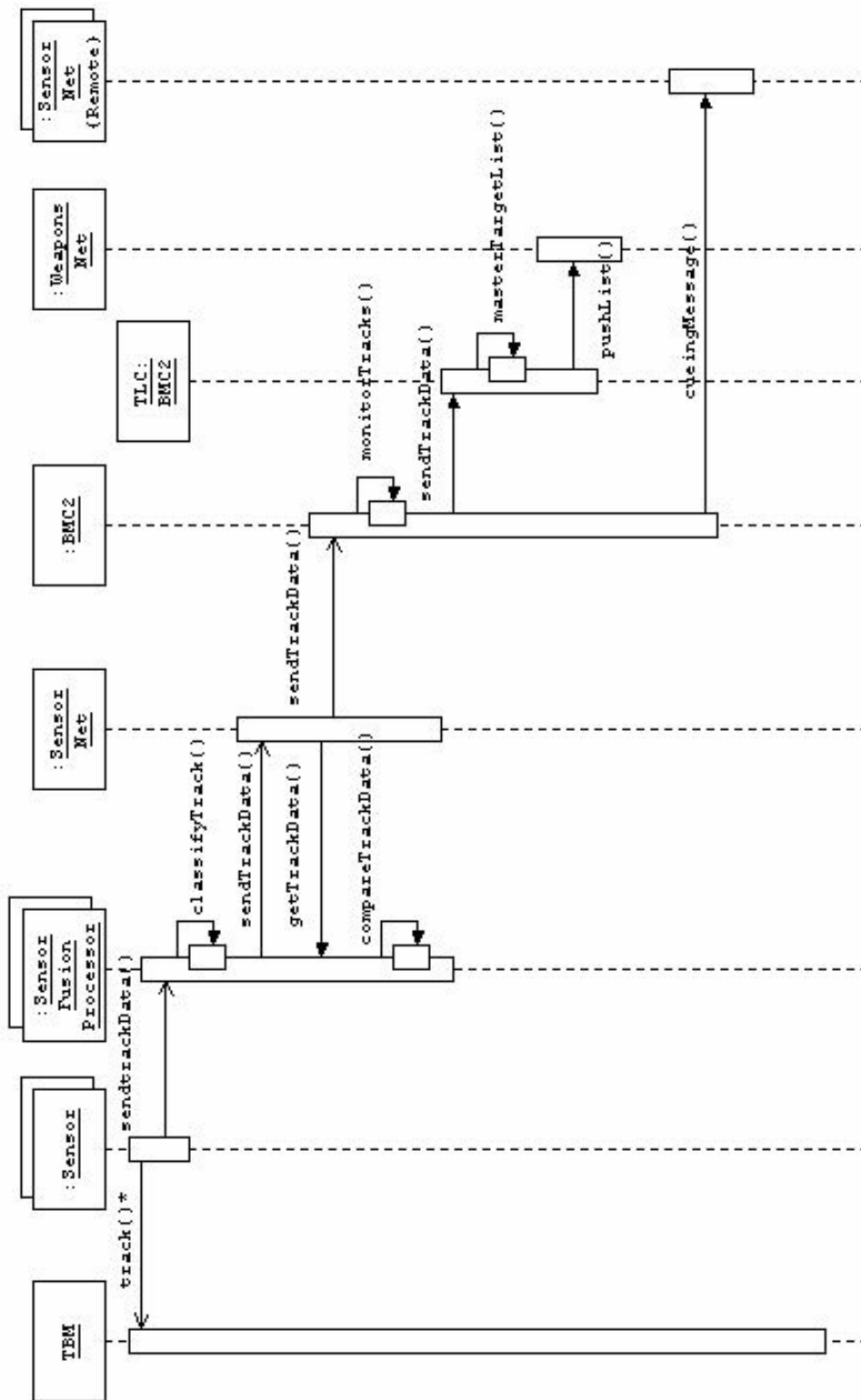


Figure 19. SSD for Use Case 2

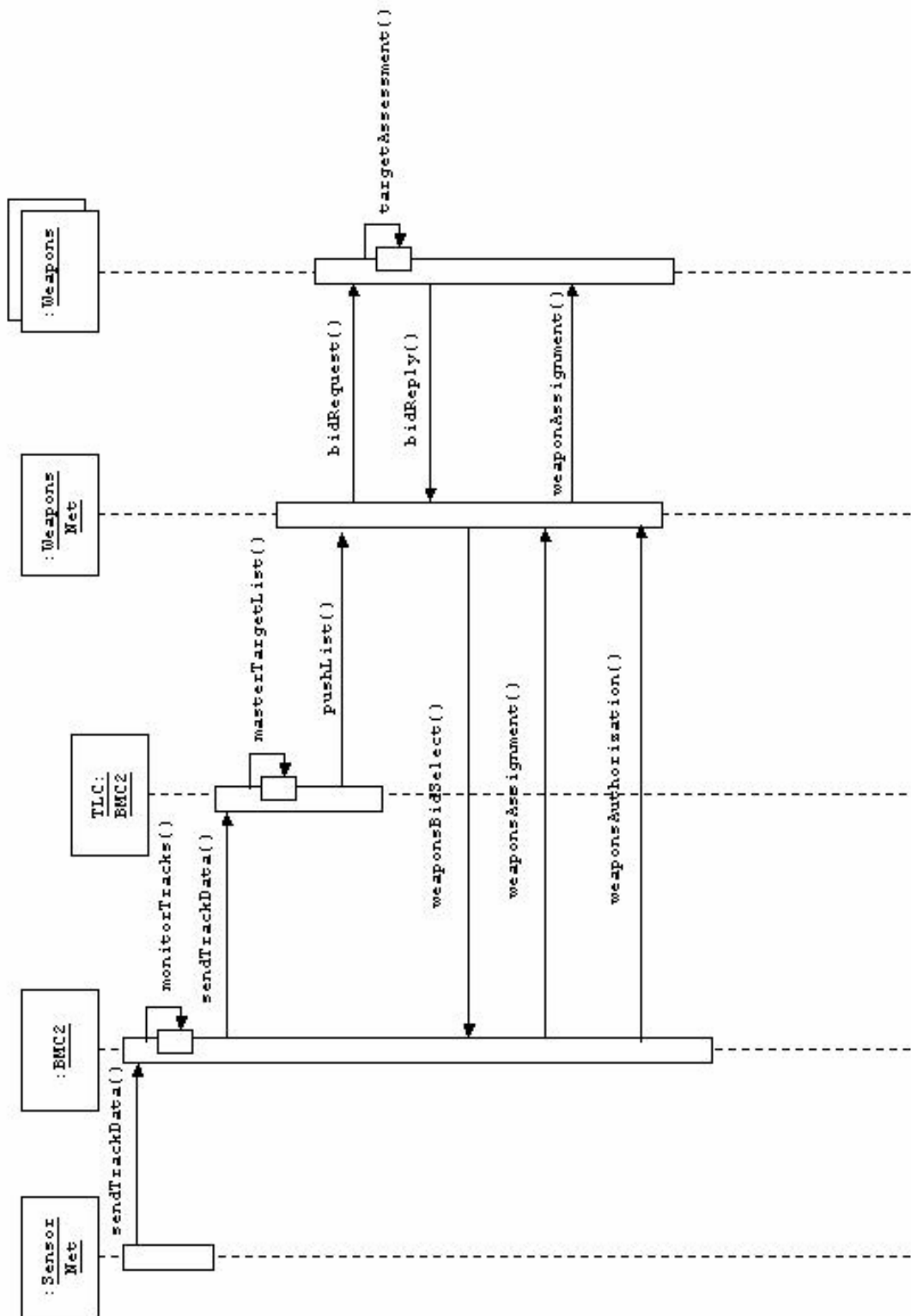


Figure 20. SSD for Use Case 3

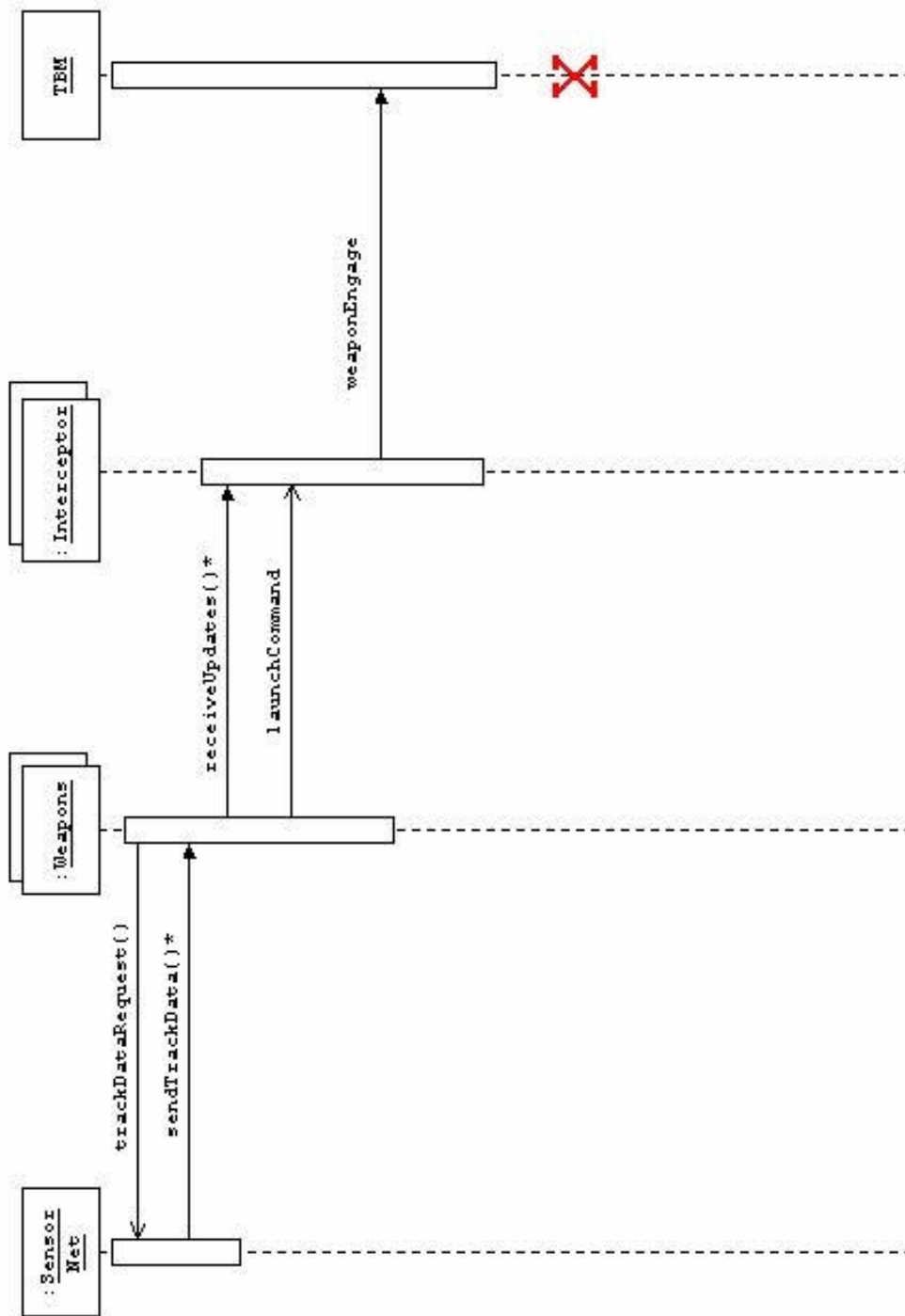


Figure 21. SSD for Use Case 4

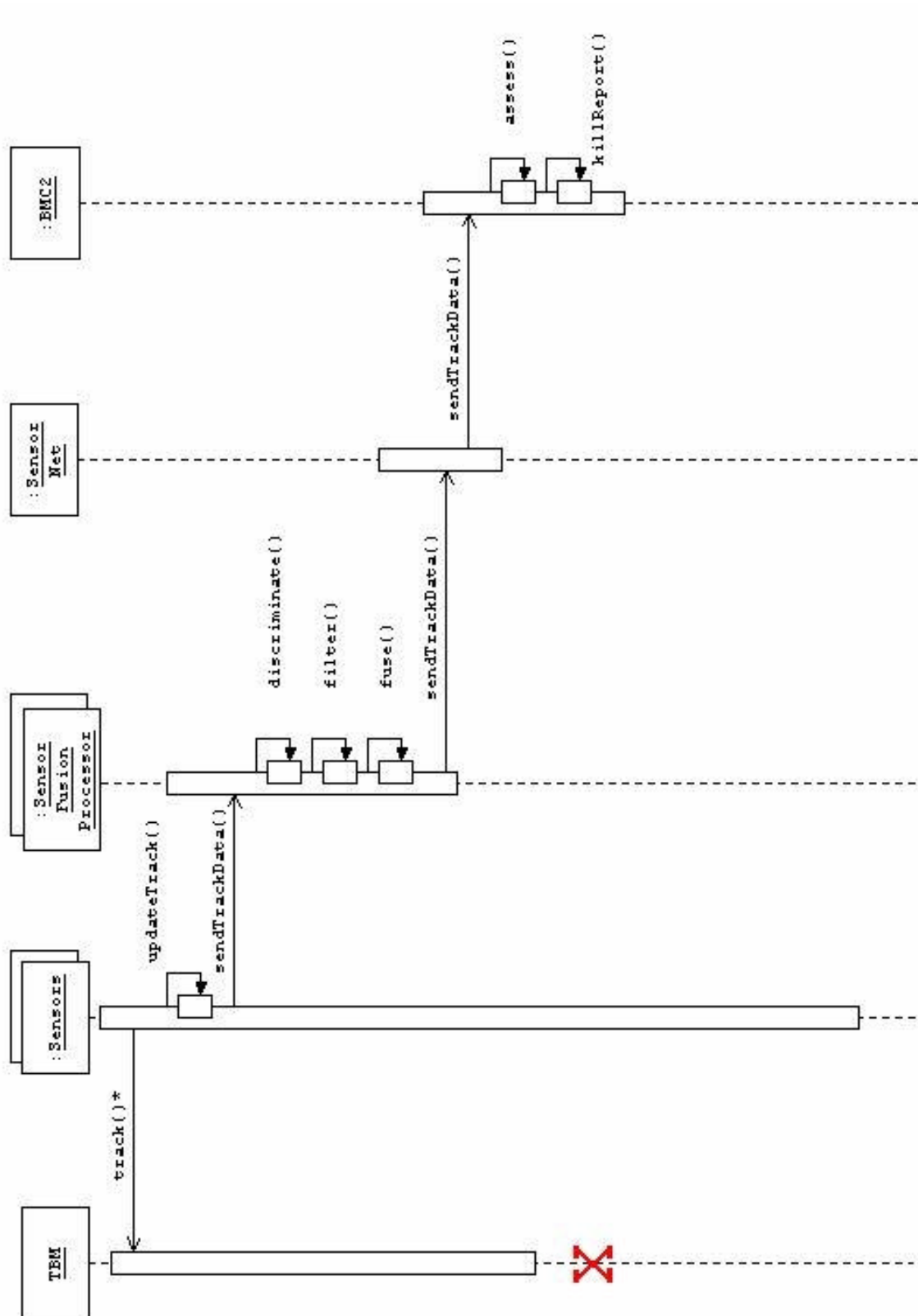


Figure 22. SSD for Use Case 5

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. UML-RT MODELS

A. SENSOR

Sensor

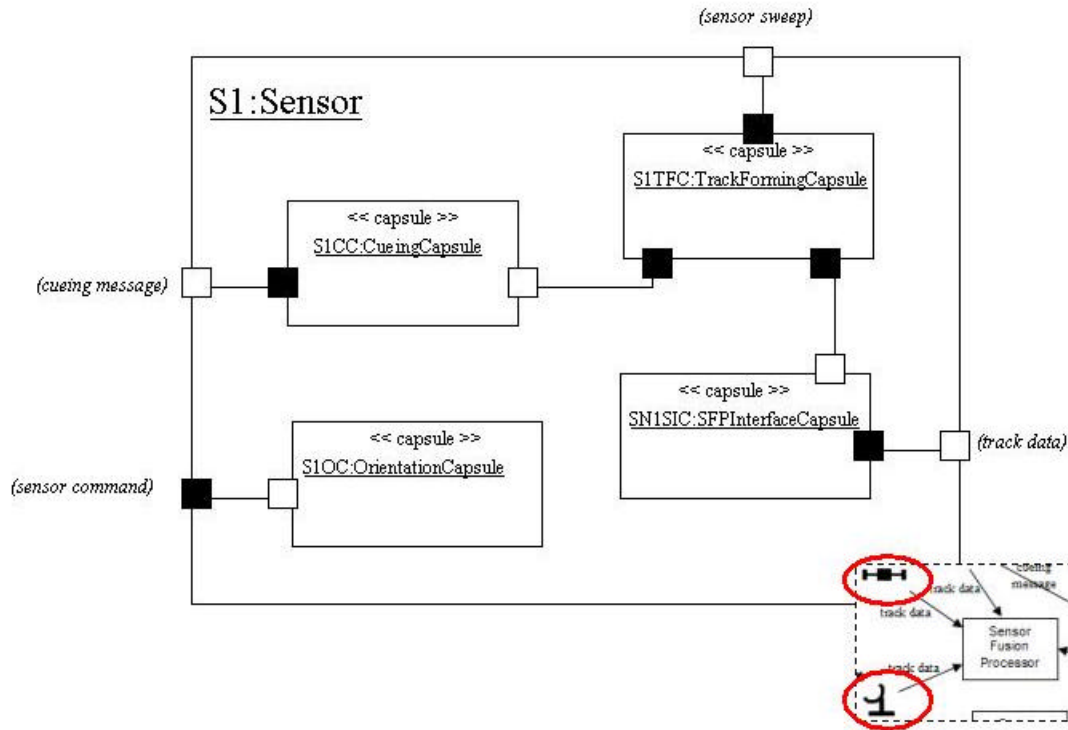


Figure 23. Sensor UML-RT Diagram

Cueing Capsule: Passes cues received by the Track Forming Capsule directly to the Sensor Net.

Orientation Capsule: Controls the orientation and scanning patterns of the sensor. It receives messages from the Sensor Controlling Command and acts on them.

Track Forming Capsule: Forms tracks from radar or IR hits. Sends Cues to the Cueing Capsule. Performs track discrimination to try to prevent debris and decoys from overloading the SFP. Once a track is adequately developed, it pushes it to the SFP Interface Capsule.

SFP Interface Capsule: Responsible for pushing tracks from the Track Forming Capsule to the sensor's higher SFP.

B. SENSOR CONTROLLING AUTHORITY

Sensor Controlling Authority

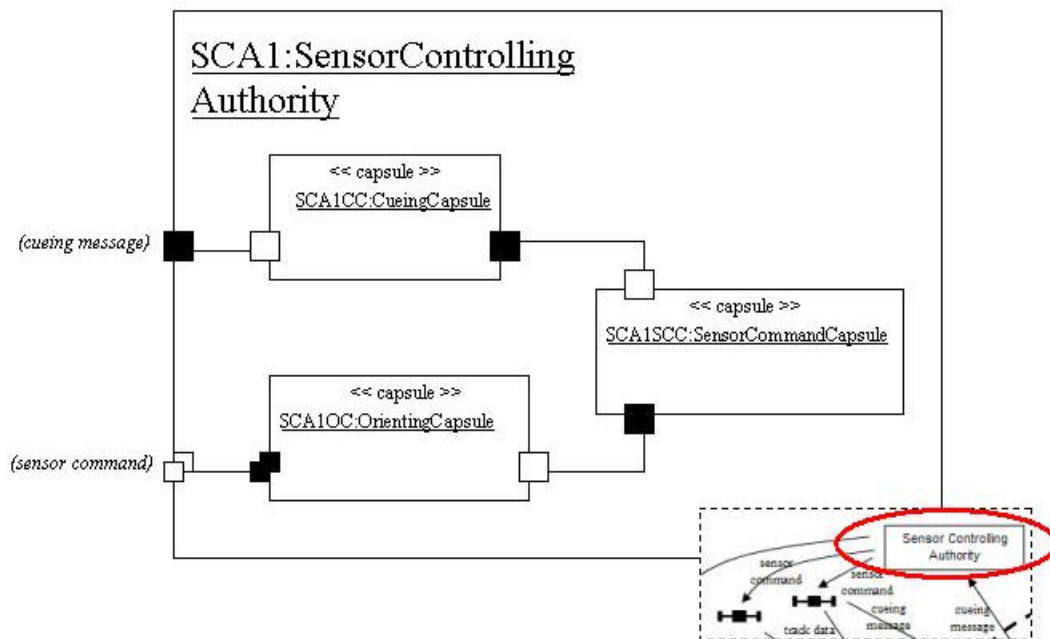


Figure 24. Sensor Controlling Authority UML-RT Diagram

Cueing Capsule: Passes cues received by Sensor Net to the Sensor Command Capsule.

Orienting Capsule: Issues commands to control the orientation and scanning patterns of subordinate sensors. It distributes commands from the Sensor Command Capsule.

Sensor Command Capsule: Receives cues from the Cueing Capsule and issues command to redirect sensor(s).

C. COMPETENT AUTHORITY

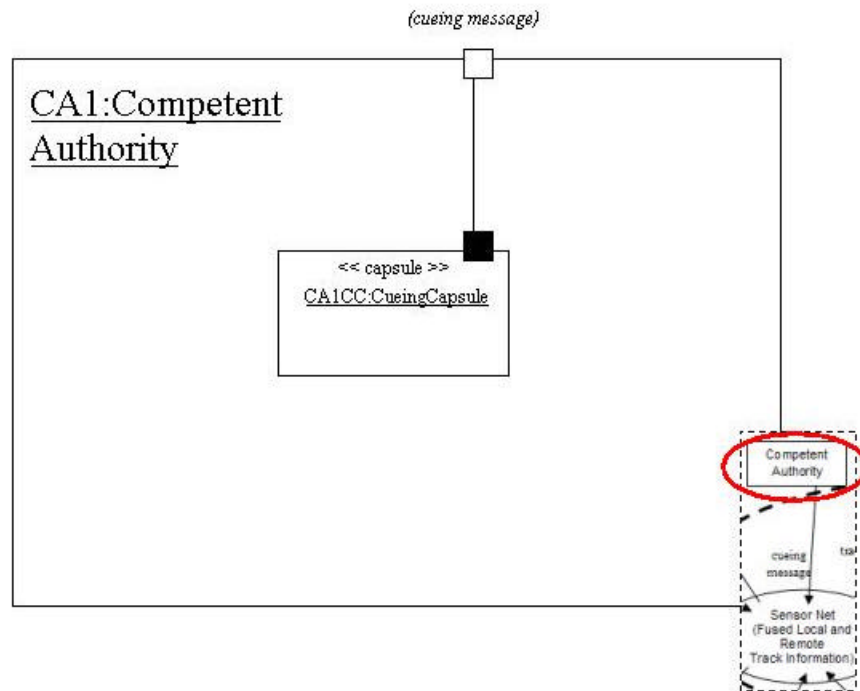


Figure 25. Competent Authority UML-RT Diagram

Cueing Capsule: Passes cues to the Sensor Net.

Track Fusing Capsule. If it is only receiving data from one sensor, then it passes it directly to the Collaborative Fusion Capsule.

Collaborative Fusion Capsule: Takes fused or raw local tracks (one per target) and fuses them with tracks received from other SFPs via the SFP Interface Capsule of the Sensor Net.

Track List Capsule: Responsible for compiling and providing the internal list of tracks for the SFP and preventing duplicates. It provides this data to both the TFC and the CFC. It provides this information upon request to the sensor net. It also serves as a repository for commands received from Sensor Net.

E. SFP'S SENSOR INTERFACE CAPSULE

Sensor Fusion Processor → Sensor Interface Capsule

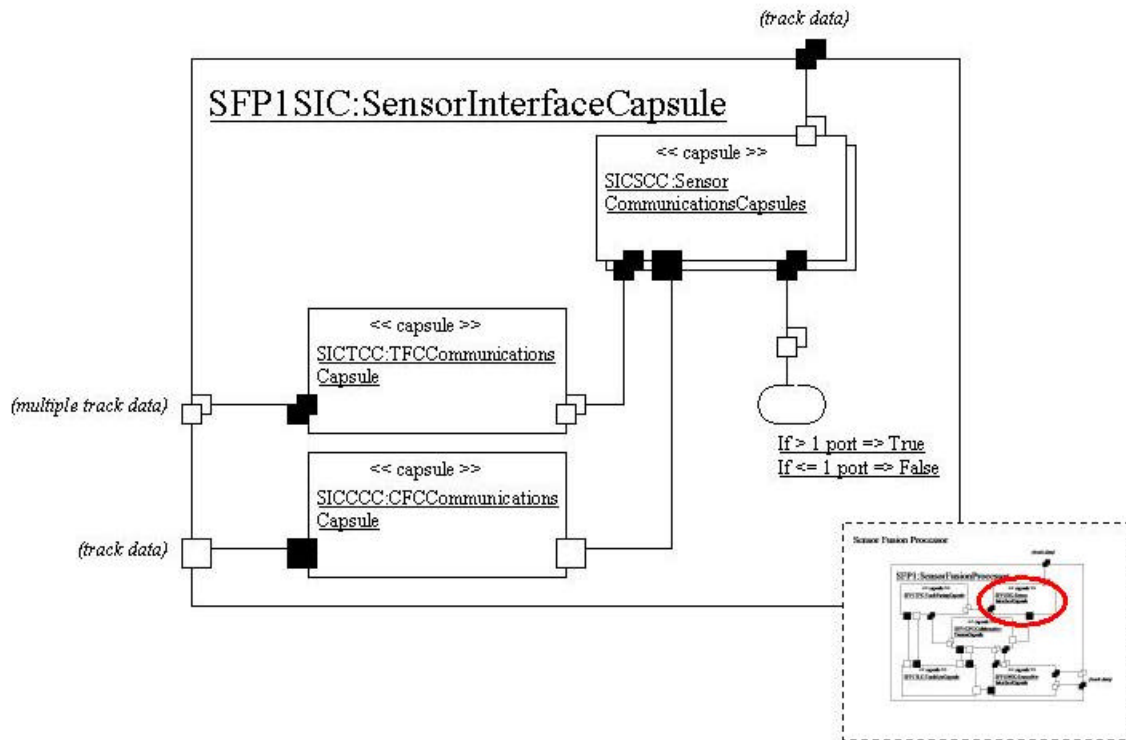


Figure 27. Sensor Interface Capsule UML-RT Diagram

Sensor Communications Capsules: Establish connections with sensors. It only allows connections from those sensors it is programmed to receive. Every time a sensor passes a track, the SCC checks the state machine to determine where to send its track data. If more than one SCC checks in a short period of time (say within 100 ms or less) then the output of the state machine turns to true. The state machine will also output true if an SCC is unable to send its data to the CFC Communications Capsule (as another sensor already has the channel tied up).

TFC Communications Capsule: Handles all data streams sent from the Sensor Communications Capsules to the (higher level) Track Fusing Capsule.

CFC Communications Capsules: Handles only one data stream sent from one Sensor Communications Capsule to the (higher level) Collaborative Fusion Capsule.

F. SFP'S TRACK FUSING CAPSULE

Sensor Fusion Processor → Track Fusing Capsule

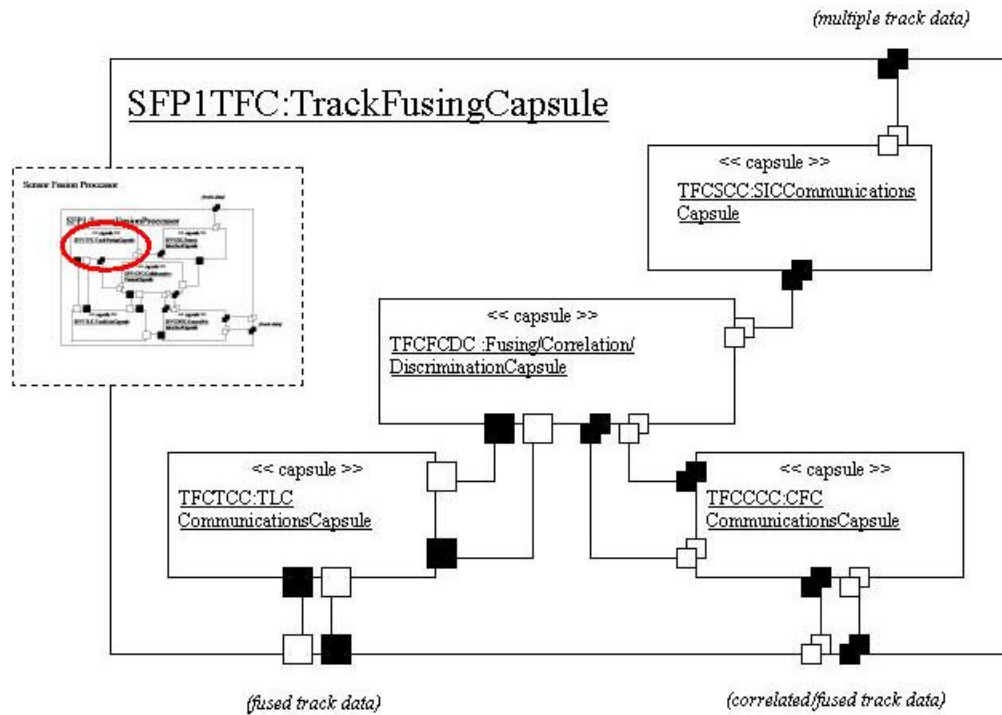


Figure 28. Track Fusing Capsule UML-RT Diagram

Fusing/Correlation/Discrimination Capsule: This capsule periodically checks each incoming track with the (higher level) Track List Capsule via the TLC Communications Capsule. If a 'stop sending' is received that pertains to the parent SFP, then the Fusing Capsule ceases dealing with that track until the 'stop sending' is lifted. Before Fusing or Correlation happens, Discrimination occurs to try to filter out the clutter and reduce the amount of tracks passed on through the rest of the Sensor Fusion Processor. Remaining tracks that pertain to targets that are new or do not have a stop order against them are then fused in real-time and forwarded to the (higher level) Collaborative Fusion Capsule.

TLC Communications Capsule: Checks all incoming tracks against the (higher level) Track List Capsule. If that track has a 'stop sending' order for the parent SFP, then the order will be passed to the Fusing/Correlation/Discrimination Capsule, thereby allowing the Fusing/Correlation/Discrimination Capsule to spend its processing power on fusing other capsules. If the track is new, it will be registered with the (higher level) Track List Capsule.

SIC Communications Capsule: Handles all data streams received from the (higher level) Sensor Interface Capsule.

CFC Communications Capsules: Handles all data streams from the Fusing Capsule sent to the (higher level) Collaborative Fusion Capsule.

G. SFP'S COLLABORATIVE FUSING CAPSULE

Sensor Fusion Processor → Collaborative Fusing Capsule

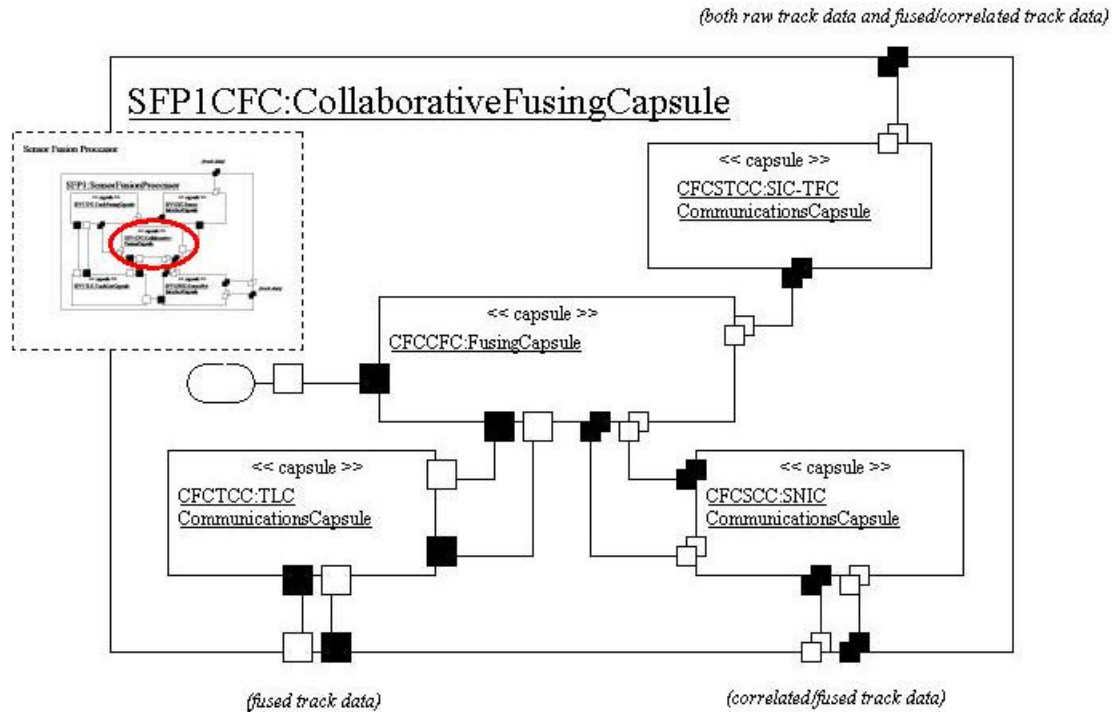


Figure 29. Collaborative Fusing Capsule UML-RT Diagram

State Machine: Contains the logic to decide whether it is worthwhile to pursue attempting to collaboratively fuse a track. This state machine contains several factors which would have to all be within acceptable parameters for it to allow collaborative fusion to occur. To reiterate in terms of a logical equation, it is an AND logic equation (for example: network usage AND track type fusible AND better fusible remote track available AND track moving slow enough AND pre-defined quality threshold not met).

Fusing Capsule: Checks each incoming track with the (higher level) Track List Capsule via the TLC Communications Capsule. If a 'stop sending' order is

received that pertains to the parent SFP, then the Fusing Capsule checks with the state machine to determine whether it should either cease dealing with that track until the 'stop sending' is rescinded or attempt to improve the track through collaborative fusion. Tracks that pertain to targets that are new or do not have a stop order against them are forwarded to the Sensor Net. If collaborative fusion is ordered for a 'stop sending' track, then the Fusing Capsule will suppress the 'stop sending' in the local (higher level) Track List Capsule. This allows the track to flow through the (higher level) Track Fusing/Correlation/Discrimination Capsule. Additionally, the Fusing Capsule will request a copy of the winning track through the Sensor Net and will fuse it with its current track. If the collaboratively fused track will beat the previous winning track, then it sends the track and continues to suppress the 'stop sending' in the local Track List Capsule until the Sensor Net feedback comes back to authoritatively either continue the 'stop sending' on the collaboratively fused track or allow it to send. If the (parent) Collaborative Fusion Capsule is receiving tracks directly from the (higher level) Sensor Interface Capsule, then the Collaborative Fusion Capsule will also perform additional discrimination as necessary. However, local fusion as performed by the (higher level) Track Fusing/Collaboration/Discrimination Capsule is of course not possible for a single sensor's input, as it requires at least two separate tracks to fuse..

TLC Communications Capsule: Checks all incoming tracks against the (higher level) Track List Capsule. If that track has a 'stop sending' order for the parent SFP, then the order will be passed to the Fusing Capsule. If

the track is new, it will register it with the (higher level) Track List Capsule.

SIC-TFC Communications Capsule: Handles all data streams received from the (higher level) Sensor Interface Capsule and Track Fusing Capsule.

SNIC Communications Capsules: Handle all data streams from the Fusing Capsule sent to the (higher level) Sensor Net Interface Capsule.

H. SFP'S TRACK LIST CAPSULE

Sensor Fusion Processor → Track List Capsule

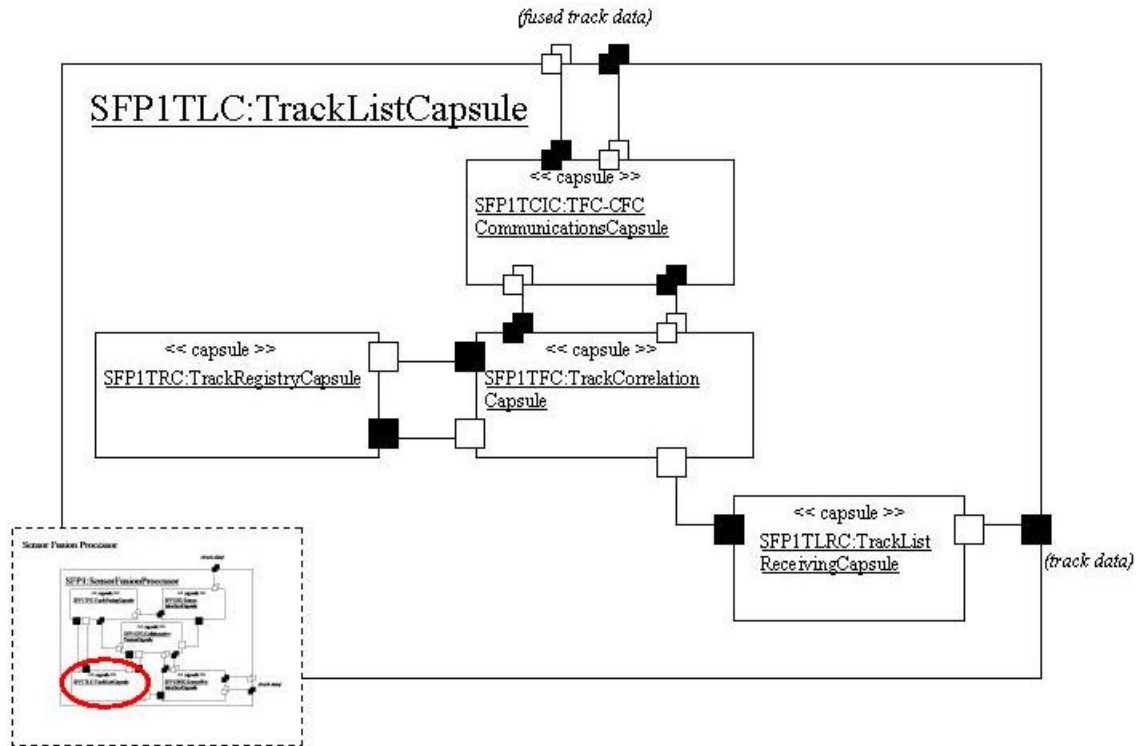


Figure 30. Track Capsule List UML-RT Diagram

TFC-CFC Communications Capsule: Handles all data streams received from the (higher level) Track Fusing/Correlation/Discrimination Capsule and Collaborative Fusing Capsule.

Track Correlation Capsule: Correlates all tracks received with Track Registry Capsule. Tracks, which are not screened out, are registered as new tracks with the Track Registry Capsule. Track numbers of tracks that are screened out are returned to the (higher level) Track Fusing/Correlation/Discrimination Capsule or (higher level) Collaborative Fusing Capsule, depending on where the track came from.

Track Registry Capsule: Maintains the SFP's master list of all perceived valid tracks as well as any additional tracks received from the Sensor Net, including any commands added to received tracks or commands pertaining to the locally maintained tracks.

Track List Receiving Capsule: Receives the Track List sent out periodically from the (higher level) Sensor Net.

I. SFP'S SENSOR NET INTERFACE CAPSULE

Sensor Fusion Processor → Sensor Net Interface Capsule

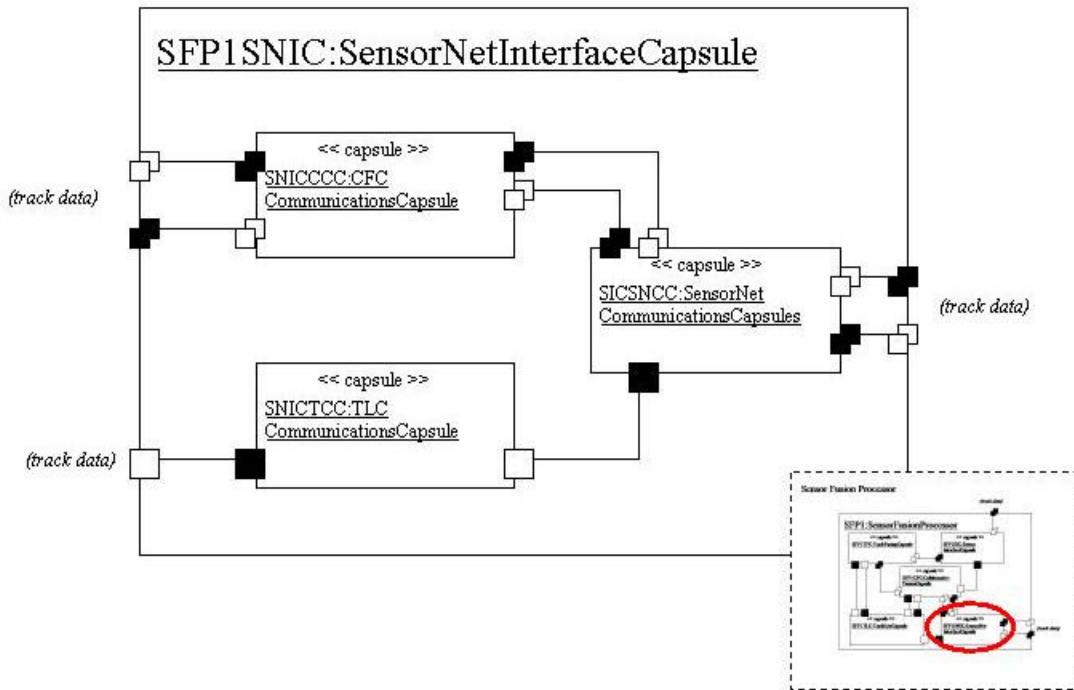


Figure 31. Sensor Net Interface Capsule UML-RT Diagram

Sensor Net Communications Capsule: Establishes connections with the Sensor Net. It handles all communications between the parent SFP and the Sensor Net.

CFC Communications Capsule: Handles all data streams between the Sensor Net Communications Capsule and the (higher level) Collaborative Fusion Capsule.

TLC Communications Capsules: Handle only one data stream sent from the Sensor Net Communications Channel to the (higher level) Track List Capsule.

J. SENSOR NET

Sensor Net

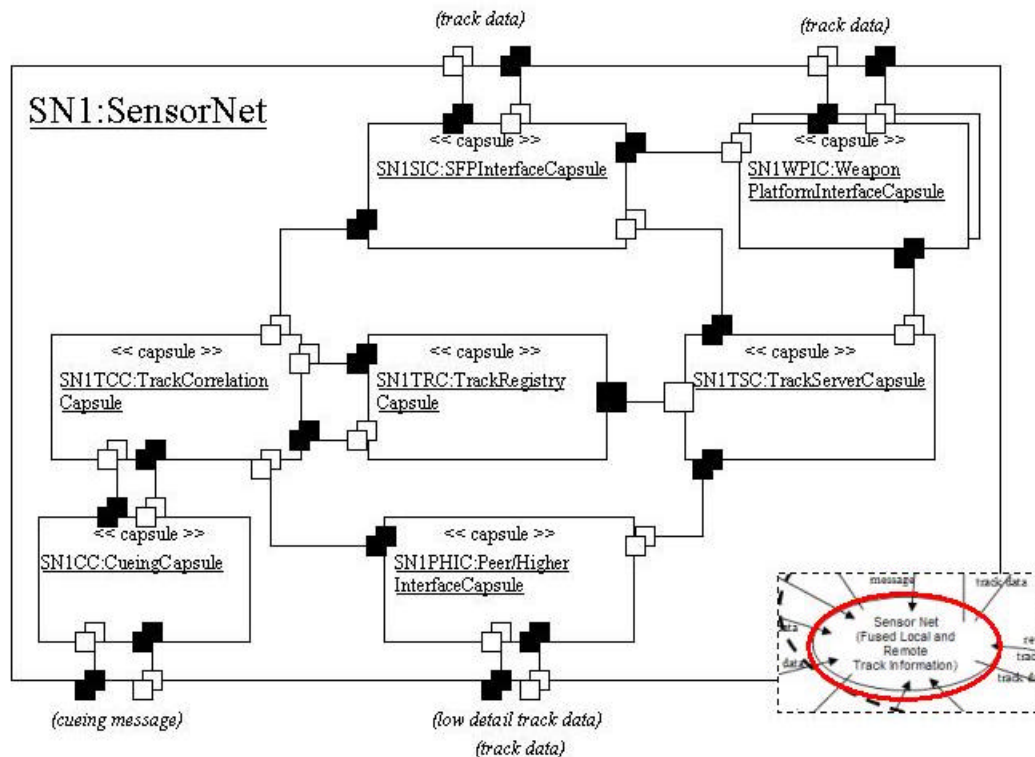


Figure 32. Sensor Net UML-RT Diagram

SFP Interface Capsule: Handles all interfaces between the SFPs and Sensor Net, including receiving fused track data, handling requests for fused data from peer SFPs, and forwarding said fused data as it is received in real time, passing track data to the Track Correlation Capsule, and receiving and forwarding track lists from the Track Server Capsule. This capsule also 'shorts' (sends before processing through the TCC/TRC/TSC loop) copies of streaming firing-solution quality data to the Weapon Platform Interface Capsule if a track is marked as 'hot' (i.e., tracks which have a weapon awaiting a firing solution to launch).

Cueing Capsule: Receives cueing messages through the Track Correlation Capsule from Trusted Sources. It checks

these cueing messages against the current Track List. If the referenced track does not correlate to a track on the list, the Cueing Capsule then passes these cueing messages to the Sensor Controlling Commands for diffusion to the various sensors.

Track Correlation Capsule: Takes all fused tracks and correlates them with the Track Registry. This correlation consists of screening out multiple instances of the same track by comparing the quality of fused tracks from multiple SFPs.

Track Registry Capsule: Maintains the SFP's master list of all perceived valid tracks. Note: All changes in a track's status are maintained ("killed," "active," "inactive," etc).

Track Server Capsule: Responsible for providing BMC2s, Weapons Net, Sensor Fusion Processors, and peer Sensor Nets with a Track List. It constantly receives an updated track list from the Track Registry Capsule and then communicates it to all requesting entities. It receives 'hot' notifications from Weapon Platforms (i.e., tracks which have a weapon awaiting a firing solution to launch) and, if it is a valid track, directs the SFP Interface Capsule to short the winning version of that track directly to the Weapon Platform Interface Capsule as well as continuing to push it to the Track Correlation Capsule.

Weapon Platform Interface Capsule: This capsule provides the latest firing-solution quality track data to requesting weapons platforms upon demand.

Peer/Higher Interface Capsule: Pushes the track list in the form of either low-detail tracks (i.e., those with no parametric data) to Peer Sensor Nets or unmodified

tracks to the BMC2. It receives their lists (peer Sensor Nets) or modifications to a Track (BMC2) and passes them to the Track Correlation Capsule for integration into the Track Registry.

Some Areas of Potential Conflict:

-What if a Weapon Platform requests a track and it is 'inactive' on the track list?

-What if a Weapon Platform comes looking for a track and it is delayed due to processing through the Sensor Net capsules? Can we ensure no delay is introduced yet firing solution quality of provided tracks is maintained?

K. SENSOR NET'S SFP INTERFACE CAPSULE

Sensor Net → SFP Interface Capsule

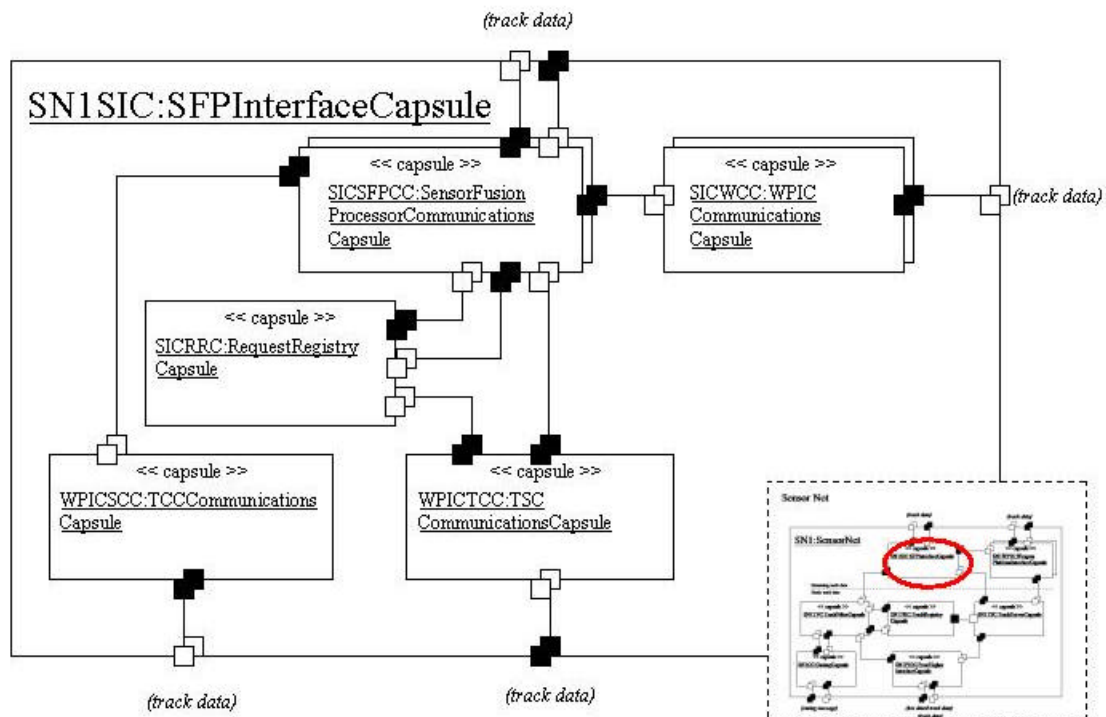


Figure 33. SFP Interface Capsule UML-RT Diagram

TSC Communications Capsule: Handles the data stream between the (higher level) Track Server Capsule and the Sensor Fusion Processor Communications Capsule. The track list and all 'short' commands from the (higher level) Track Registry Capsule are passed to the Request Registry Capsule.

TCC Communications Capsule: Passes tracks from the Sensor Fusion Processor Communications Capsule to the (higher level) Track Correlation Capsule.

WPIC Communications Capsules: Passes 'shorted' tracks from the Sensor Fusion Processor Communications Capsule to the (higher level) Weapon Platform Interface Capsule.

Request Registry Capsule: Maintains a registry of all incoming and outgoing data streams as well as a current copy of the track list from the (higher level) Track Registry. This capsule is responsible for retrieving and forwarding track data to SFPs that want to enhance their own tracks with remote SFP's track data (this makes the SFP Forum concept into an 1 X N bandwidth usage structure instead of an N X N bandwidth usage structure, which is what happens if you have SFPs talking to one another). The Request Registry Capsule does not automatically retrieve and forward data requested by the various SFPs, however. It has an internal state machine that decides whether or not to pass it. This internal state machine is identical to the SFP's Collaborative Fusing Capsule's equation, with the exception that the RRC's state machine has a better view of the network and its components. The state machine's decision to either allow or disallow a collaborative fusion request to be processed can be understood to be the result of an evaluation of several key factors, all of which must be within allowable parameters. This state machine

decision is most accurately understood as an AND logic equation (network usage AND track type fusible AND better fusible remote track available AND track moving slow enough AND pre-determined quality threshold not met). This capsule also receives all 'short' commands and ensures that the appropriate data stream is 'shorted' to the WPIC Communications Capsule with the appropriate track ID on the data stream.

Sensor Fusion Processor Communications Capsules:
Establish communications between Sensor Fusion Processors and the Sensor Net. Handle all data flows between the SFPs and the Sensor Net. Passes tracks to the Request Registry Capsule when so directed by the RRC.

L. SENSOR NET'S TRACK FILTER CAPSULE

Sensor Net → Track Correlation Capsule

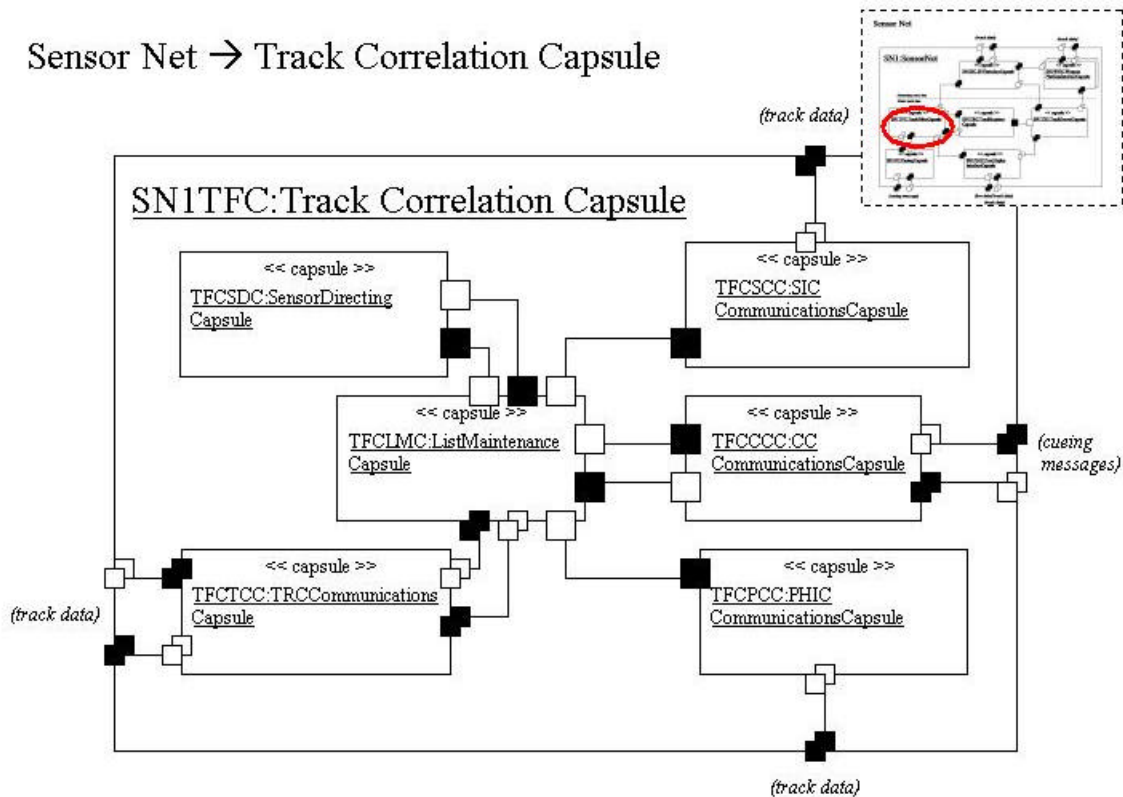


Figure 34. Track Filter Capsule UML-RT Diagram

Sensor Directing Capsule: Contains the logic that decides what to do with tracks as they come in. Tracks that are correlated out have a 'stop sending' order added to the track data, which will be put on the master track list and distributed to the parent SFP for action.

List Maintenance Capsule: The concept of this capsule is that it must take these vast streams of data flowing into it and, after comparing them to the list of current tracks, decide which of these constitute new tracks that must be added to the list or which are better tracks than what was previously had, and which are to be dropped. This processing is all done in parallel without the benefit of mutual exclusion. Therefore, in order to ensure accurate decisions are tacked onto the tracks before they are passed to the (higher level) Track Registry Capsule, it then serializes all input and passes it along with its decisions to the Sensor Directing Capsule so that the tracks can receive appropriate markings to cause the correct actions to be taken by the SFPs.

CC Communications Capsule: Receives cues from the (higher level) Cueing Capsule and passes them to the List Maintenance Capsule. Unlike all others, it receives a valid/invalid response directly from the List Maintenance Capsule before the LMC writes to the (higher level) Track Registry Capsule.

PHIC Communications Capsule: Handles all traffic from the (higher level) Peer/Higher Interface Capsule. It takes potentially multiple feeds and serializes them to reduce the timing complexity.

SIC Communications Capsule: Handles all traffic from the (higher level) SFP Interface Capsule. It takes potentially multiple feeds and serializes them.

TRC Communications Capsule: Forwards all tracks with embedded commands to the (higher level) Track Registry Capsule. It also forwards requests for the master track list from the List Maintenance Capsule and replies with the master track list to the List Maintenance Capsule once the master track list is received.

M. SENSOR NET'S CUEING CAPSULE

Sensor Net → Cueing Capsule

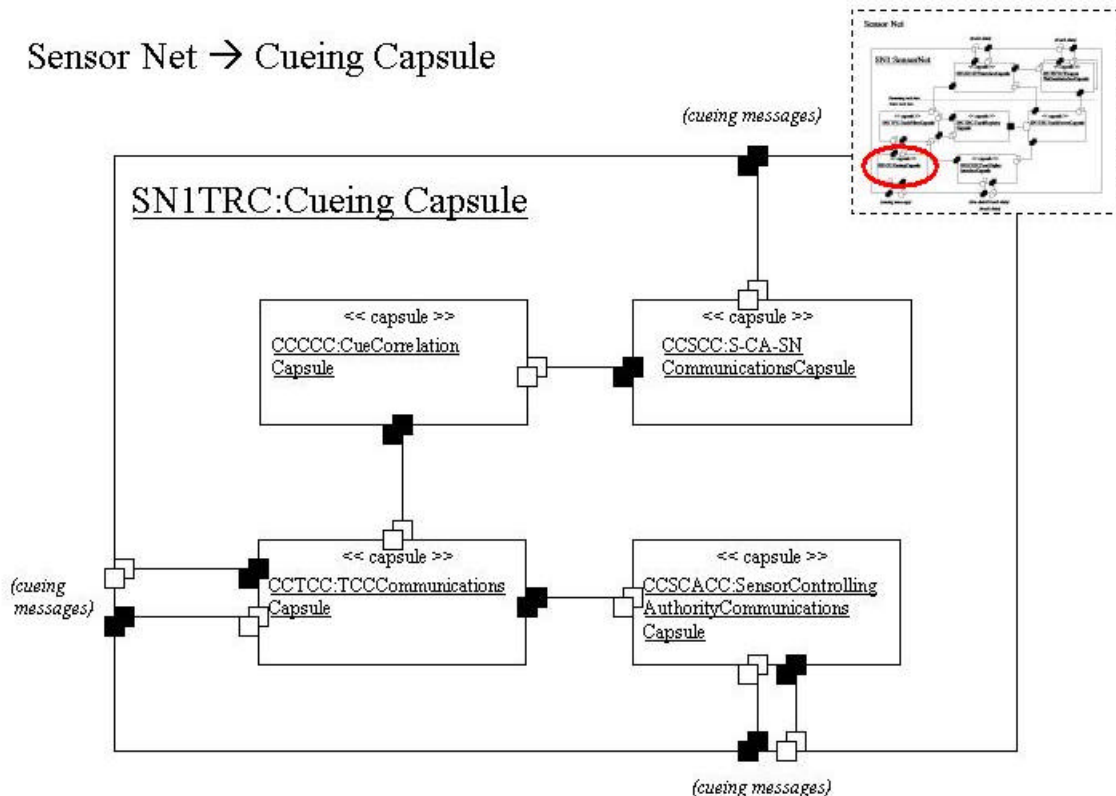


Figure 35. Cueing Capsule UML-RT Diagram

Cue Correlation Capsule: This capsule screens all inputs received using a list of the last several seconds of cues. If it determines that a cue is virtually the same as one in its resident memory, then it will drop the cue rather than forward it. Those that pass screening are forwarded to the TFC Communications Capsule.

S-CA-SN Communications Capsule: Receives cues directly from Sensors, Competent Authorities, and other Sensor Nets and forwards them to the Cue Correlation Capsule.

TCC Communications Capsule: Forwards all cues to the (higher level) Track Correlation Capsule. It then receives a copy of each cue back with either 'invalid' or 'valid' on it. It forwards the 'valid' cues to the Sensor Controlling Authority Communications Capsule for dissemination.

Sensor Controlling Authority Communications Capsule: responsible for establishing communications with Sensor Controlling Authorities. It forwards all valid cues to all Sensor Controlling Authorities affiliated with the Sensor Net.

Note: The whole cueing system is a separate system from the track list passing that goes on through other channels. It is designed to be much faster to allow Sensor Controlling Authorities the maximum decision time possible.

N. SENSOR NET'S TRACK REGISTRY CAPSULE

Sensor Net → Track Registry Capsule

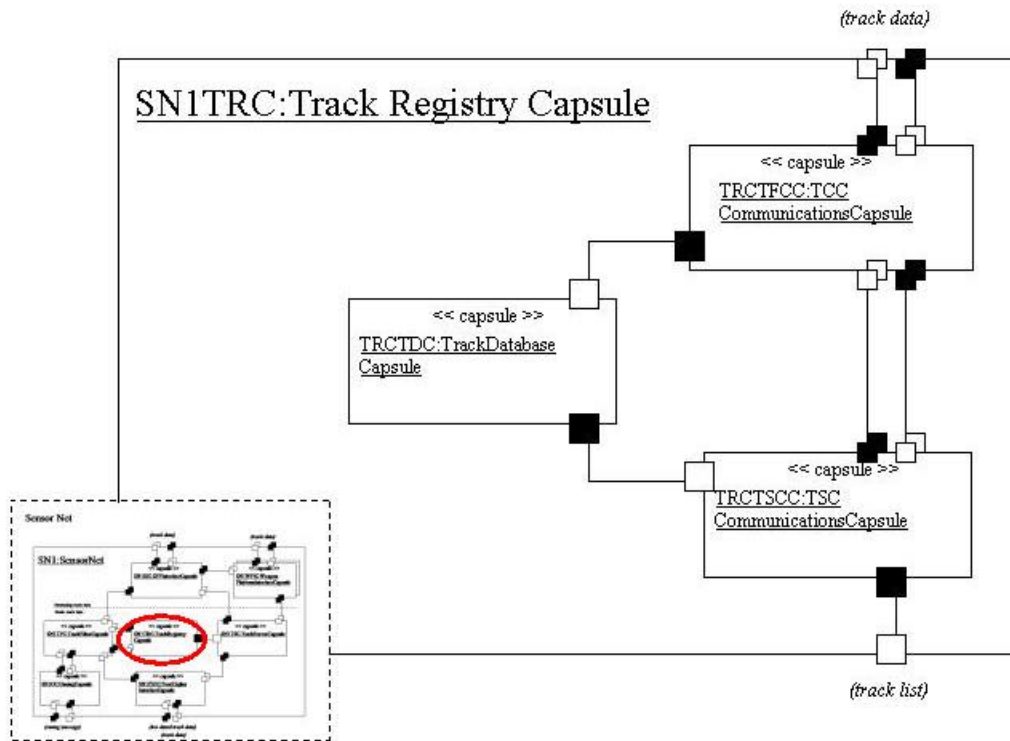


Figure 36. Track Registry Capsule UML-RT Diagram

Track Database Capsule: Maintains the Sensor Net's master list of all perceived valid tracks. Note: All changes in a track's status are maintained ("killed," "active," "inactive," etc).

TCC Communications Capsule: Handles all data streams between the (higher level) Track Communications Capsule and the Track Database and TSC Communications Capsules. It passes valid, post-correlation tracks to the Track Database Capsule for writing to the database. It passes the (higher level) Track Correlation Capsule's requests for a copy of the master track list to the TSC Communications Capsule and receives that list. It then passes the master track list to the (higher level) Track Correlation Capsule.

TSC Communications Capsule: Receives the master track list from the Track Database Capsule and distributes it to the (higher level) Track Server Capsule as well as the TCC Communications Capsule.

O. SENSOR NET'S TRACK SERVER CAPSULE

Sensor Net → Track Server Capsule

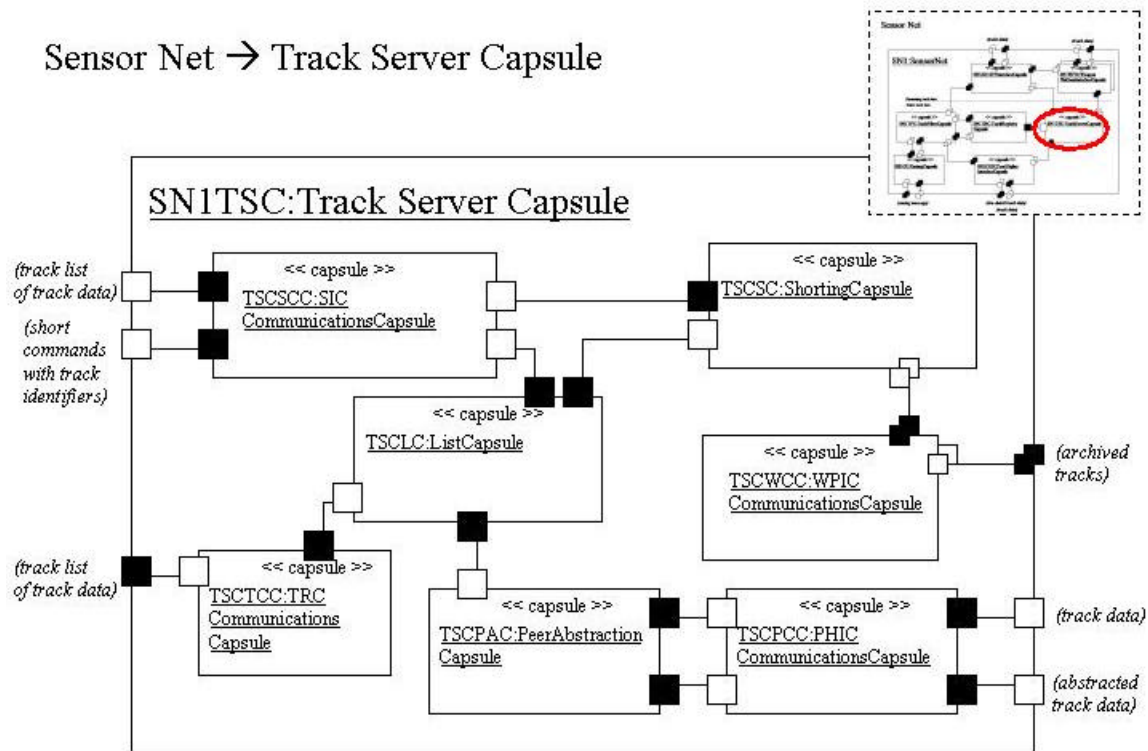


Figure 37. Track Server Capsule UML-RT Diagram

Shorting Capsule: Receives requests for streaming telemetry data from the WPIC Communications Capsule. It then matches it to a track from the most current track list and sends a 'short order' to the SIC Communications Capsule.

List Capsule: Receives the track list from the TRC Communications Capsule and streams it out to all connected capsules.

Peer Abstraction Capsule: Receives the track list from the List Capsule and streams two copies to the PHIC Communications Capsule. The first copy is unmodified and is meant for the BMC2. The second copy gets abstracted (i.e., unnecessary detail is removed) and is meant for Peer Sensor Nets.

WPIC Communications Capsule: Receives requests for streaming telemetry data from the Weapon Platform Communications Capsules and passes these requests to the Shorting Capsule, passing back an acknowledgement to the weapon when the request has been shorted or passing back some other status if not able to comply.

PHIC Communications Capsule: Receives data from the Peer Abstraction Capsule and passes it via one of two ports (depending on whether or not it is abstracted) to the (higher level) Peer/Higher Interface Capsule.

TRC Communications Capsule: Receives the track list from the (higher level) Track Registry Capsule and passes it to the List Capsule.

SIC Communications Capsule: Receives the track list from the List Capsule and Shorting Orders from the Shorting Capsule. Forwards them to the (higher level) SFP Interface Capsule for action.

P. SENSOR NET'S PEER/HIGHER INTERFACE CAPSULE

Sensor Net → Peer/Higher Interface Capsule

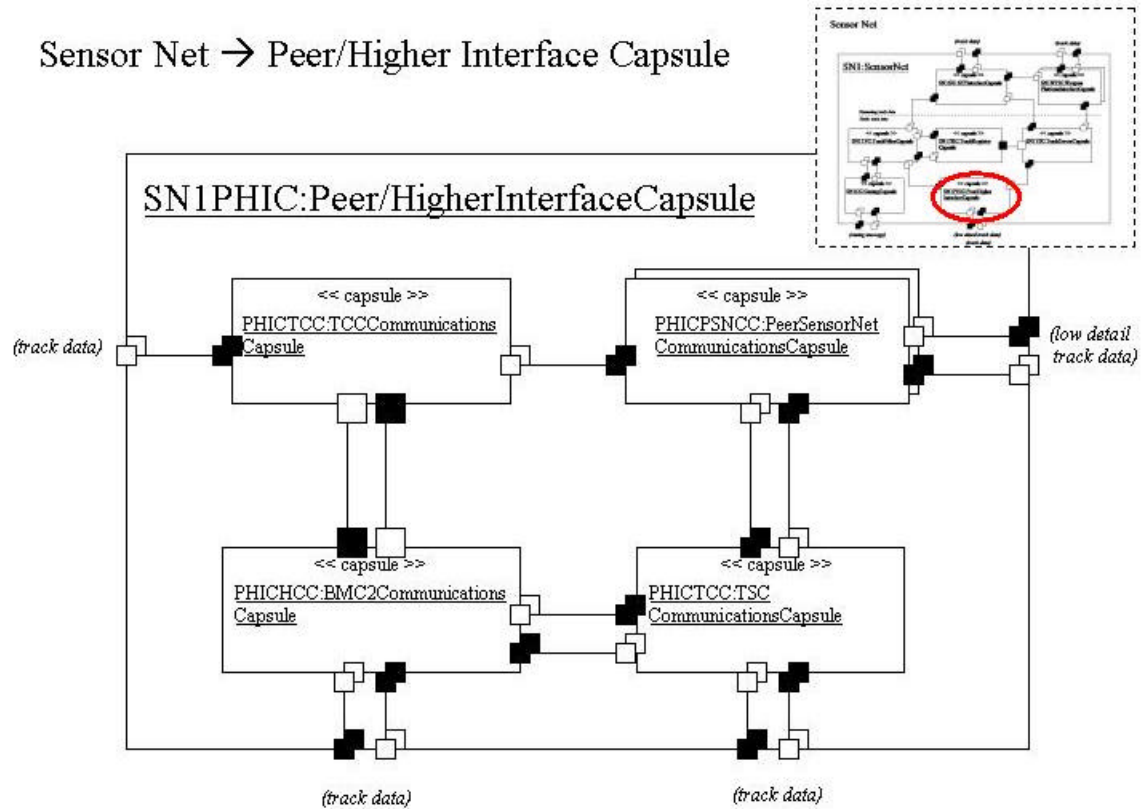


Figure 38. Peer/Higher Interface Capsule UML-RT Diagram

TSC Communications Capsule: Handles all data streams between the BMC2 Communications Capsule or Peer Sensor Net Communications Capsule and the (higher level) Track Server Capsule.

TCC Communications Capsule: Receives modifications from the BMC2 Communications Capsule as well as Low Detail Track Lists from the Peer Sensor Net Communications Capsule and passes it to the (higher level) Track Correlation Capsule.

Peer Sensor Net Communications Capsules: Establish communications between the owning Sensor Net and its peer Sensor Nets. They then handle requests by peer sensor nets for Track Lists of Low Detail Track Data, passing such

TSC Communications Capsule: Handles all data streams between the Weapon Platform Communications Capsule and the (higher level) Track Server Capsule.

SIC Communications Capsules: Receive shorted tracks from the (higher level) SFP Interface Capsule. These tracks are passed to the appropriate Weapon Platform Communications Capsule.

Weapon Platform Communications Capsules: Establishes communications between the Sensor Net and Weapon Platforms. Each instantiation of this capsule carries a priority, which is the priority of the target the weapon platform is assigned (obtained from the BMC2's target list by the weapon platform and is part of the request for a firing solution).

R. WEAPONS PLATFORM

Weapon Platform

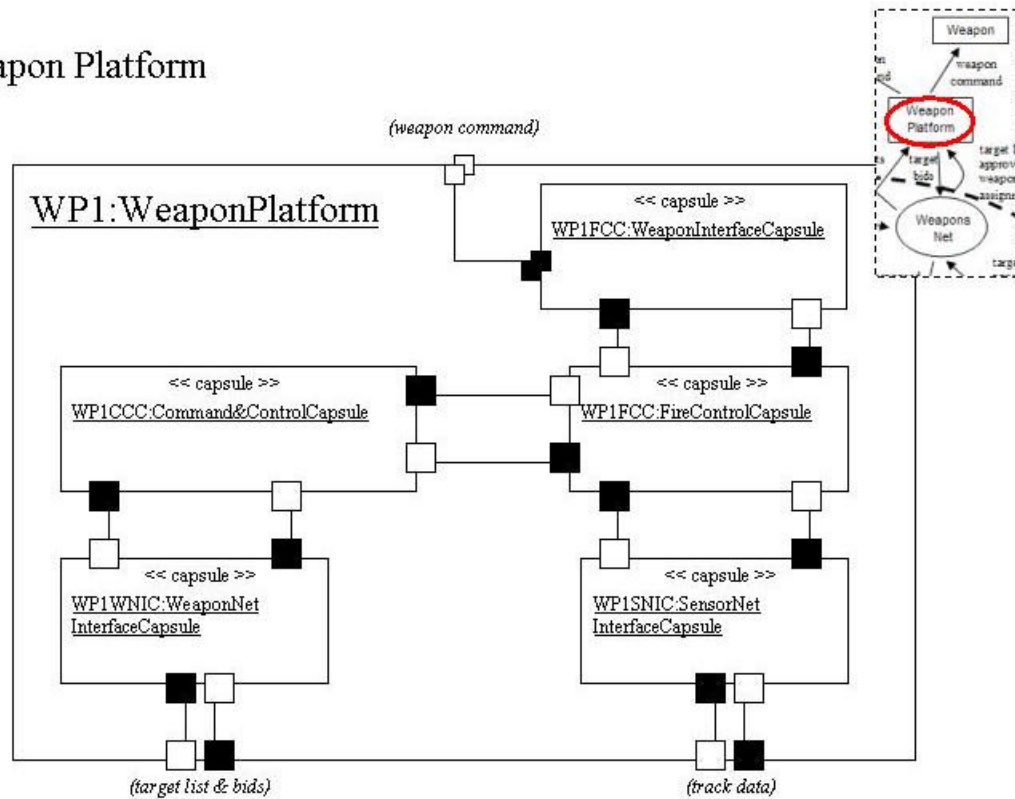


Figure 40. Weapons Platform UML-RT Diagram

Weapon Interface Capsule: Relays fire-control data to the weapons and performs all weapon interface functions.

Command & Control (C2) Capsule: Receives Target List from Weapon Net Interface Capsule. It then uses the Fire Control Capsule to generate its bids, then submits the bids through the WNIC to the Weapon Net. This capsule also oversees the Fire Control Capsule and issues commands to its own weapons (through the FCC).

Fire Control Capsule: Performs all normal fire control functions, computes target bids, and requests track data through the SNIC.

Sensor Net Interface Capsule: Requests and relays track information for assigned targets.

Weapon Net Interface Capsule: Receives and replies to Target List Bid Requests. Relays assigned tracks to C2 capsule.

S. BMC2

BMC2

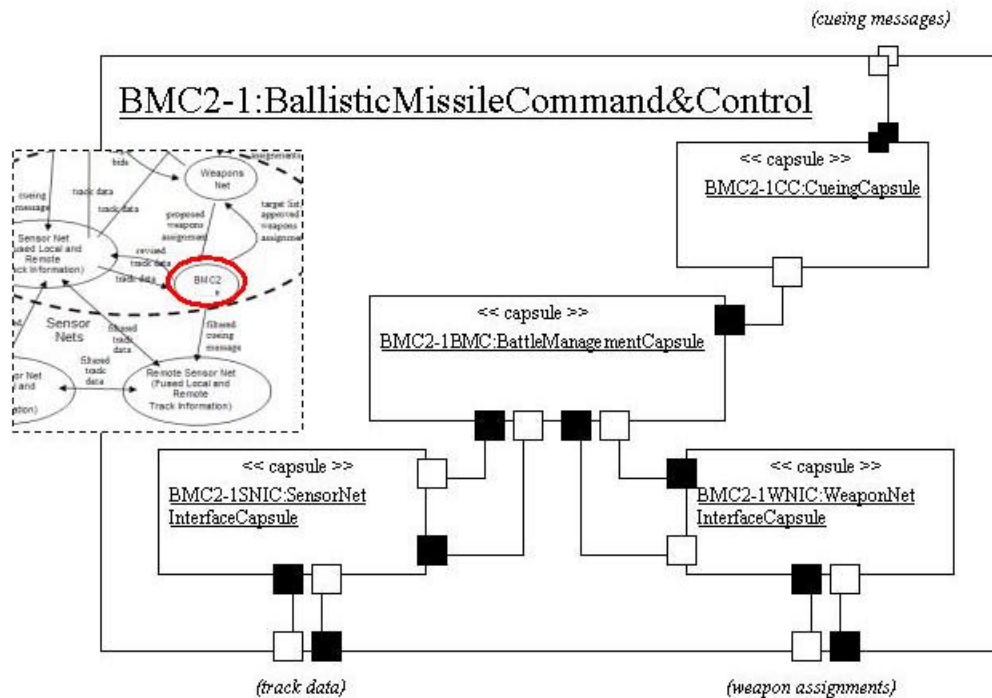


Figure 41. BMC2 UML-RT Diagram

Cueing Capsule: Passes abstracted track data as cueing messages to peer Sensor Nets.

Battle Management Capsule: Validates or modifies proposed weapons assignments from Weapons Net. Controls and updates Master Target List. Does Predictive Tracking for current tracks.

Sensor Net Interface Capsule: Receives Track List from Sensor Net and pushes modifications due to C2 Overrides back to Sensor Net.

Weapon Net Interface Capsule: Pushes the Master Target List to Weapon Net and receives the proposed weapons assignments.

T. WEAPON

Weapon

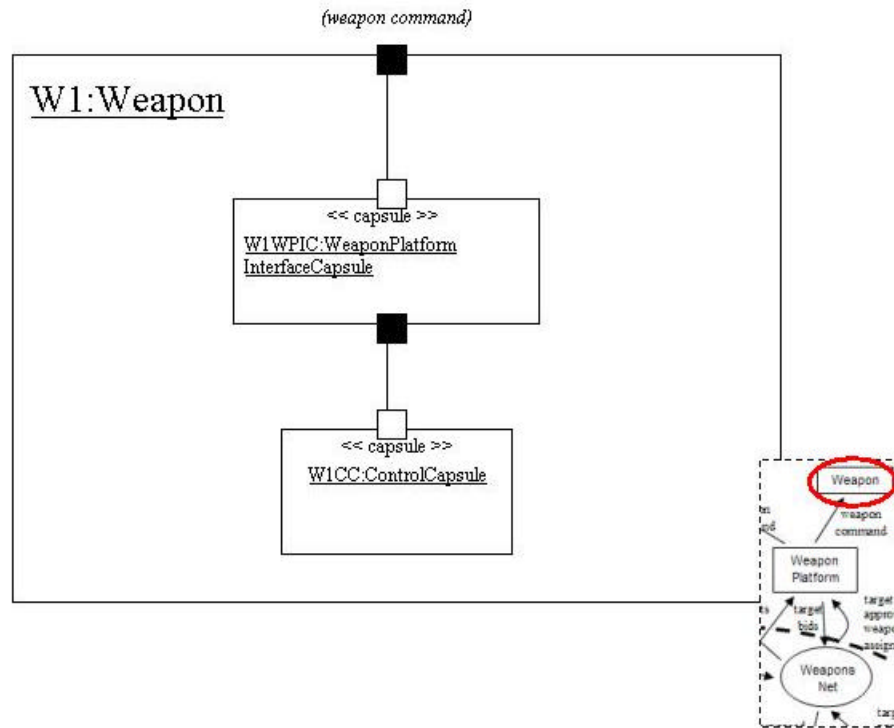


Figure 42. Weapon UML-RT Diagram

Control Capsule: Receives guidance and telemetry data through WPIC from parent Weapon Platform. Controls the weapon and provides feedback to the weapon platform.

Weapon Platform Interface Capsule: Establishes and maintains communications with the parent Weapon Platform.

U. WEAPON NET

Weapon Net

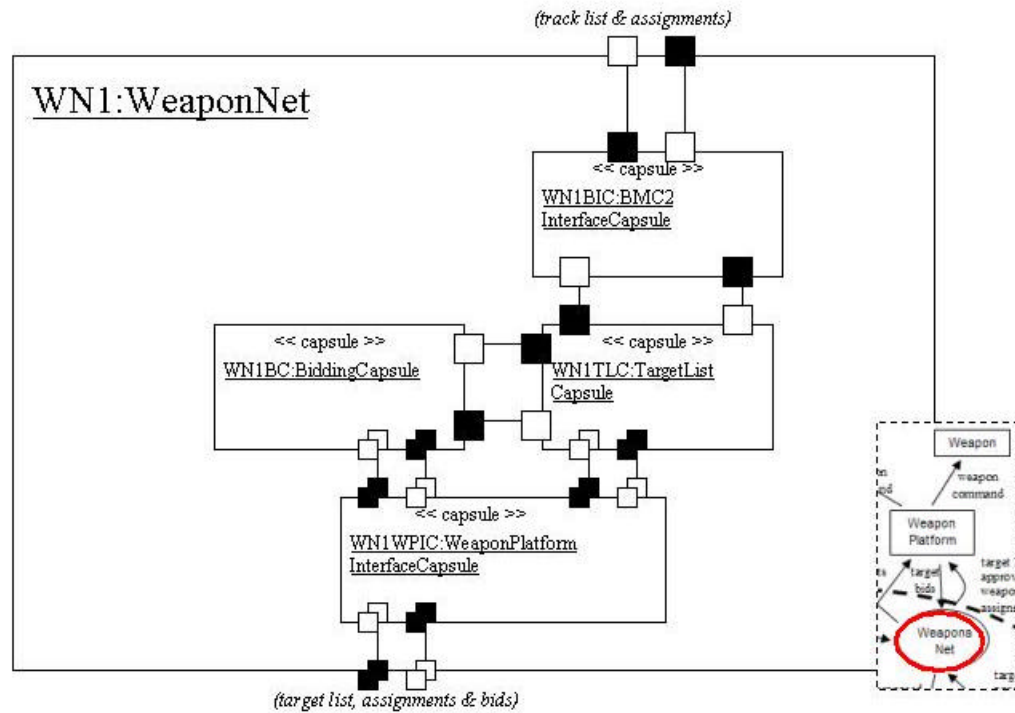


Figure 43. Weapon Net UML-RT Diagram

Control Capsule: Receives guidance and telemetry data through WPIC from parent Weapon Platform. Controls the weapon and provides feedback to the weapon platform.

Weapon Platform Interface Capsule: Establishes and maintains communications with the parent Weapon Platform.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F. SIMULATION CODE

A. SFP SIMULATION CODE.

Read Me File.

Notes on this Simulation:

OMNeT++ does not handle 0 modules of a type being instantiated. Therefore, you must instantiate at least one of both types of sensors. Because of this, it is impossible to test the SIC to CFC link (which is only used when there's one sensor). The simulation would have to be retooled, and one of the two sensors would have to be removed.

In the Plove analysis, it will always appear that ColFus Tracks get to SensorNet faster than Normal Tracks. This is not true. It appears this way because the radar sensors are labeled starting with 0, whereas the numbers of the IR Sensors follow after the radar sensors (so if there's 3 radar and 2 IR, your radar will be 0,1,2, and your IR will be 3,4). IR has a natural delay of 500ms in it (mathematical equation based on orbital distance, speed of light, and a 93000Hz satellite downlink frequency). Tracks are chosen for fusion starting with sensor 0 and going up to the number of collaborative fusion requests - 1. Therefore, radar sensors will always be chosen for fusion before IR sensors, which inadvertently ensures that at least some of the Normal Tracks will have a 500ms delay. This kicks the average throughput time for Normal Tracks up above those of Collaboratively Fused Tracks.

Time Constraints were gathered from the following:

Process_Time = Estimate from Professor Wen Su based on simple routing of the message at the IP layer with no packet analysis other than source and destination.

ListCheck = Estimate from Professor Wen Su based on basic XORing function which XORs the most significant bits (gets it within an ellipse of certainty) from an associative memory bank containing a master track list of a couple of hundred items.

Fusion = total guess as sensor fusion has yet to be invented.

Track Size for both IR and Radar are based on a summation of fields that would be required for a space ballistic missile tracks. We added in additional bits for any

additional system overhead, etc, that may not have existed in other data link systems such as JTIDS.

```
//-----  
----
```

```
// file: SFPSim.ned  
// author: Joel D. Babbitt  
// Thesis Work @ NPS  
// Date: 14 Nov 2003
```

```
//-----  
----
```

```
// RadarSensor --
```

```
//
```

```
// A ground based radar sensor which sends sensor data to  
the SFP.
```

```
//
```

```
simple RadarSensor
```

```
    gates:
```

```
        out: out;
```

```
endsimple
```

```
// IRSensor --
```

```
//
```

```
// A satellite based IR sensor which sends sensor data to  
the SFP.
```

```
//
```

```
simple IRSensor
```

```
    gates:
```

```
        out: out;
```

```
endsimple
```

```
// SensorInterfaceCapsule --
```

```
// Serves as the primary interface to all assigned Sensors.
```

```

// If it is receiving data from more than one sensor, then
it sends all tracks

// to the Track Fusing Capsule. If it is only receiving
data from one sensor,

// then it passes it directly to the Collaborative Fusion
Capsule.

//
simple SensorInterfaceCapsule
    gates:
        in: in[]; // in from multiple sensors
        out: TFCout[]; // out to Track Fusing Capsule
(multiple connections)
        out: CFCout; // out to Collaborative Fusion Capsule
(only one connection)
endsimple

// SensorNetInterfaceCapsule --
//
// Responsible for pushing tracks from the Track List
Capsule to the Sensor Net.
// Receives tracks requested by the Collaborative Fusion
Capsule through Sensor
// Net from other SFPs.
//
simple SensorNetInterfaceCapsule
    gates:
        out: SNRequestout[]; //port used to request
collaboratively fused tracks from SN
        in: SNRequestin[]; //port used to receive
collaboratively fused tracks from SN
        out: CFCRequestout[]; //port used to pass
collaboratively fused tracks to CFC
        in: CFCRequestin[]; //port used to receive requests
from CFC for c. fused tracks
        in: TrackListin; //port used to receive the master
track list from SN

```

```

        out: TrackListout; //port used to push the master
track list to the TLC
        in: CFCin[]; //port used to receive tracks from CFC
        out: SNout[]; //port used to push tracks to SN
endsimple

```

```

// TrackFusingCapsule --
//
// Takes multiple tracks per target from the Sensor
Interface Capsule and fuses
// them into one single track per target in real time.
//
simple TrackFusingCapsule
    gates:
        in: SICin[]; //port that receives tracks from SIC
        out: CFCout[]; //port used to push tracks to CFC
        out: TLCout[]; //port used to check target list
        in: TLCin[]; //port used to receive answers from
TLC
endsimple

```

```

// CollaborativeFusionCapsule --
//
// Takes fused or raw local tracks (one per target) and
fuses them with tracks
// received from other SFPs via the SFP Interface Capsule
of the Sensor Net.
//
simple CollaborativeFusionCapsule
    gates:
        in: SICin; //port that receives tracks from SIC
        in: TFCin[]; //port that receives tracks from TFC
        out: SNICout[]; //port used to push tracks to SNIC
        out: TLCout[]; //port used to check target list

```

```

        in: TLCin[]; //port used to receive answers from
TLC
        out: SNICRequestout[]; //port used to request c.
fused tracks from SN
        in: SNICRequestin[]; //port used to receive c.
fused tracks from SN
endsimple

```

```

//TrackListCapsule --
//
// References its internal track list, meshes the master
track list with its own.
//
simple TrackListCapsule
    gates:
        in: TrackListin; //port that receives the master
target list from the SNIC
        in: TFCin[]; //port that receives tracks to be
checked from the TFC
        out: TFCout[]; //port used to reply to the TFC's
queries
        in: CFCin[]; //port that receives tracks to be
checked from the CFC
        out: CFCout[]; //port used to reply to the CFC's
queries
endsimple

```

```

// SensorNet --
//
// Pushes master track list to the SFP, receives tracks
from the SFP, and handles
// requests for tracks from other SFPs.
//
simple SensorNet
    gates:

```

```

        out: SFPRequestout[]; //port used to push
collaboratively fused tracks to the SFP

        in: SFPRequestin[]; //port used to receive c. fused
track requests from SFP

        out: TrackListout; //port used to push the master
track list to the TLC

        in: SFPin[]; //port used to receive tracks from SFP
endsimple

```

```

// SFPSim --

```

```

//

```

```

// Model of the Sensor Fusion Processor, with connections
to multiple sensors and

```

```

// one Sensor Net.

```

```

//

```

```

module SFPSim

```

```

    parameters:

```

```

        //parameters that involve only one entity

```

```

        data_rate_RadarSensorToSFP : numeric, // the data
rate between Radar Sensor and the SFP

```

```

        RadarTrackSize : numeric, // size of an unfused
radar track

```

```

        RadarTrackDelay : numeric const, // delay between
radar tracks being sent to the SFP

```

```

        data_rate_IRSensorToSFP : numeric, // the data rate
between IR Sensor and the SFP

```

```

        IRTrackDelay : numeric const, // delay between IR
tracks being sent to the SFP

```

```

        IRTrackSize : numeric, // size of an unfused IR
track

```

```

        //parameters that involve more than one entity

```

```

        num_RadarSensors : numeric, // the number of Radar
Sensors

```

```

        num_IRSensors : numeric, // the number of IR
Sensors

```

```

        num_Tracks : numeric, //the number of real tracks
out there (ie: planes, rockets, missiles, etc)

        data_rate_SFPTtoSensorNet : numeric, // the data
rate between the SFP and SensorNet

        data_rate_Internal : numeric, // data rate of
connections within the SFP

        TrackListDelay : numeric, //amount of delay between
sendings of the master track list

        num_FusionRequests : numeric, // number of
collaborative fusion requests from CFC (<=num_Tracks)

        FusedTrackSize : numeric, // size of a firing
solution quality fused track

        Process_Time : numeric, // Generic handling time
each module eats in handling a track

        ListCheck : numeric, // Time Required to Check a
Track against the List

        Fusion : numeric; //Time Required to perform a
Fusing Action

        submodules:

            TrackFusingCapsule: TrackFusingCapsule;

            gatesizes:

                SICin[num_RadarSensors+num_IRSensors],
//port that receives tracks from SIC

                CFCout[num_Tracks], //port used to push
tracks to CFC

                TLCout[num_RadarSensors+num_IRSensors],
//port used to check target list

                TLCin[num_RadarSensors+num_IRSensors];
//port used to receive answers from TLC

                display: "p=79,59,r,70;i=comp;b=36,32";

            CollaborativeFusionCapsule:
CollaborativeFusionCapsule;

            gatesizes:

                SICin, //port that receives tracks from SIC

                TFCin[num_Tracks], //port that receives
tracks from TFC

                SNICout[num_Tracks], //port used to push
tracks to SNIC

```

```

        TLCout[num_Tracks], //port used to check
target list
        TLCin[num_Tracks], //port used to receive
answers from TLC
        SNICRequestout[num_FusionRequests], //port
used to request c. fused tracks from SN
        SNICRequestin[num_FusionRequests]; //port
used to receive c. fused tracks from SN
        display: "p=136,155,r,70;i=comp;b=36,32";
        TrackListCapsule: TrackListCapsule;
        gatesizes:
                TrackListin, //port that receives the
master target list from the SNIC
                TFCin[num_RadarSensors+num_IRSensors],
//port that receives tracks to be checked from the TFC
                TFCout[num_RadarSensors+num_IRSensors],
//port used to reply to the TFC's queries
                CFCin[num_Tracks], //port that receives
tracks to be checked from the CFC
                CFCout[num_Tracks]; //port used to reply to
the CFC's queries
        display: "p=69,284,r,70;i=comp;b=36,32";
        SensorNet: SensorNet;
        gatesizes:
                SFPRequestout[num_FusionRequests], //port
used to push collaboratively fused tracks to the SFP
                SFPRequestin[num_FusionRequests], //port
used to receive c. fused track requests from SFP
                TrackListout, //port used to push the
master track list to the TLC
                SFPin[num_Tracks]; //port used to receive
tracks from SFP
        display: "p=375,291;i=router3;b=36,32";
        SensorNetInterfaceCapsule:
SensorNetInterfaceCapsule;
        gatesizes:
                SNRequestout[num_FusionRequests], //port
used to request collaboratively fused tracks from SN

```



```

        SNRequestin[num_FusionRequests],      //port
used to receive collaboratively fused tracks from SN

        CFCRequestout[num_FusionRequests],    //port
used to pass collaboratively fused tracks to CFC

        CFCRequestin[num_FusionRequests],     //port
used to receive requests from CFC for c. fused tracks

        TrackListin, //port used to receive the
master track list from SN

        TrackListout, //port used to push the
master track list to the TLC

        CFCin[num_Tracks], //port used to receive
tracks from CFC

        SNout[num_Tracks]; //port used to push
tracks to SN

        display: "p=224,284;i=router;b=32,32";
        IRSensor: IRSensor[num_IRSensors]; //
        display:
        "p=395,69,r,90;i=satellitesensoricon;b=75,97";
        SensorInterfaceCapsule: SensorInterfaceCapsule;
        gatesizes:
        TFCout[num_RadarSensors+num_IRSensors],
        CFCout,
        in[num_RadarSensors+num_IRSensors];
        display: "p=227,60,r,80;i=router;b=32,32";
        RadarSensor: RadarSensor[num_RadarSensors];
        display:
        "p=393,196,r,100;i=radarsensoricon;b=92,88";
        //see p43 of the manual for figuring out problems with
ports (especially the [] parts)
        connections:
        //connect up the Radar Sensors to the SIC
        for i=0..num_RadarSensors-1 do
            RadarSensor[i].out    -->    delay    5ms    -->
SensorInterfaceCapsule.in[i];
        endfor;
        //connect up the IR Sensors to the SIC also
        for i=0..num_IRSensors-1 do

```

```

        IRSensor[i].out    -->    delay    500ms    -->
SensorInterfaceCapsule.in[num_RadarSensors + i];
    endfor;
    //connect up the SIC to the TFC
    for i=0..(num_RadarSensors+num_IRSensors)-1 do
        SensorInterfaceCapsule.TFCout[i] --> delay 0ms
--> TrackFusingCapsule.SICin[i];
    endfor;
    //connect up the SIC to the CFC
    SensorInterfaceCapsule.CFCout --> delay 0ms -->
CollaborativeFusionCapsule.SICin;

    //connect up the TFC with the TLC
    for i=0..(num_RadarSensors+num_IRSensors)-1 do
        TrackFusingCapsule.TLCout[i] --> delay 0ms -->
TrackListCapsule.TFCin[i];
    endfor;
    for i=0..(num_RadarSensors+num_IRSensors)-1 do
        TrackFusingCapsule.TLCin[i] <-- delay 0ms <--
TrackListCapsule.TFCout[i];
    endfor;
    //connect up the TFC with the CFC
    for i=0..num_Tracks-1 do
        TrackFusingCapsule.CFCout[i] --> delay 0ms -->
CollaborativeFusionCapsule.TFCin[i];
    endfor;

    //connect up the CFC with the TLC
    for i=0..num_Tracks-1 do
        CollaborativeFusionCapsule.TLCout[i] --> delay
0ms --> TrackListCapsule.CFCin[i];
    endfor;
    for i=0..num_Tracks-1 do
        CollaborativeFusionCapsule.TLCin[i] <-- delay
0ms <-- TrackListCapsule.CFCout[i];
    endfor;

```

```

//connect up the CFC with the SNIC
for i=0..num_FusionRequests-1 do
    CollaborativeFusionCapsule.SNICRequestout[i] --
> delay 0ms --> SensorNetInterfaceCapsule.CFCRequestin[i];
endfor;
for i=0..num_FusionRequests-1 do
    CollaborativeFusionCapsule.SNICRequestin[i] <--
delay 0ms <-- SensorNetInterfaceCapsule.CFCRequestout[i];
endfor;
for i=0..num_Tracks-1 do
    CollaborativeFusionCapsule.SNICout[i] --> delay
0ms --> SensorNetInterfaceCapsule.CFCin[i];
endfor;

//connect up the SNIC with the SensorNet and TLC to pass
the master track list through
//all delays are based on a dedicated T-1 or faster line.
    SensorNetInterfaceCapsule.TrackListin <-- delay 5ms
<-- SensorNet.TrackListout;
    SensorNetInterfaceCapsule.TrackListout --> delay
5ms --> TrackListCapsule.TrackListin;
    //connect up the SNIC with the SensorNet to request
collaboratively fused tracks
    for i=0..num_FusionRequests-1 do
        SensorNetInterfaceCapsule.SNRequestout[i] -->
delay 5ms --> SensorNet.SFPRequestin[i];
    endfor;
    for i=0..num_FusionRequests-1 do
        SensorNetInterfaceCapsule.SNRequestin[i] <--
delay 5ms <-- SensorNet.SFPRequestout[i];
    endfor;
    //connect up the SNIC with the SensorNet to pass
tracks to SensorNet
    for i=0..num_Tracks-1 do
        SensorNetInterfaceCapsule.SNout[i] --> delay
5ms --> SensorNet.SFPin[i];
    endfor;

```

```

        display: "p=18,18;b=273,301";
endmodule

//
// Instantiates a Sensor Fusion Processor.
//
network TheSFPSim : SFPSim // must match file name (e.g.
test.ned)

    parameters:

        num_RadarSensors = input(4,                "Number
of Ground-Based Radar Sensors:_____"),

        num_IRSensors = input(2,                  "Number
of Satellite-Based IR Sensors:_____"),

        num_Tracks = input (5,                    "Number
of actual objects being tracked:_____"),

        data_rate_SFPTtoSensorNet = input(45000000, "Data
Rate (bps) between SFP and SensorNet:_____"),

        data_rate_Internal = input(10000000000,    "Data
Rate (bps) between Capsules:_____"),

        FusedTrackSize = input(500,                "Size
(bits) of Fused Track:_____"),

        data_rate_RadarSensorToSFP = input(45000000,"Data
Rate (bps) between Radar Sensor and the SFP:___"),

        RadarTrackSize = input(500,                "Size
(bits) of an Unfused Radar Track:_____"),

        RadarTrackDelay = input(.5,                "Delay
(sec) between Radar Tracks sent to the SFP:___"),

        data_rate_IRSensorToSFP = input(93000,     "Data
Rate (bps) between IR Sensor and the SFP:_____"),

        IRTrackDelay = input (2,                    "Delay
(sec) between IR tracks being sent to the SFP:___"),

        IRTrackSize = input (500,                    "Size
(bits) of an unfused IR track:_____"),

        TrackListDelay = input(.1,                "Delay
(sec) between Master Track List broadcasts:___"),

        num_FusionRequests = input(1,              "Number
of collaborative fusion requests from CFC:___"),

```

```

        Process_Time = input(.000005,
        "Time (sec) each Module takes to handle a
track:_____"),
        ListCheck = input(.0005, "Time
(sec) to check a track against the List:_____"),
        Fusion = input(.01,
        "Time (sec) required to perform a Fusing
Action:_____");

```

```
endnetwork
```

```

//-----
// file: RadarSensor.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 14 Nov 2003
// This is a generic radar sensor.
//-----

```

```
#include "omnetpp.h"
```

```

class RadarSensor : public cSimpleModule
{
    Module_Class_Members(RadarSensor,cSimpleModule,16384)
    virtual void activity();
};
Define_Module( RadarSensor );
void RadarSensor::activity()
{
    int own_addr = gate( "out" )->toGate()->index();
    int track_size = parentModule()->par("RadarTrackSize");
    int num_tracks = parentModule()->par("num_Tracks");
    double delay = parentModule()->par("RadarTrackDelay");
    bool sim_start = true;
    for(;;)
    {

```

```

        if (!sim_start)
        {
            // keep an interval between batches of tracks
            being sent out
            wait( delay );
        }
        sim_start = false;
        for(int i=0;i<num_tracks; i++) //send out one
        track per object out there.
        {
            // connection setup
            ev << "Client " << name() << " " << own_addr
            << " sending Radar Track of size " << track_size << "
            bits\n";

            cMessage *work = new cMessage( name());
            work->addPar("src") = own_addr;
            work->addPar("fwd") = true;
            work->setLength(track_size);
            work->setTimestamp(); //puts a current
            time timestamp on it.
            send( work, "out" );
        }
    }
}

```

```

//-----
// file: IRSensor.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 14 Nov 2003
// This is a generic IR sensor.
//-----

```

```

#include "omnetpp.h"

```

```

class IRSensor : public cSimpleModule

```

```

{
    Module_Class_Members(IRSensor,cSimpleModule,16384)
    virtual void activity();
};

Define_Module( IRSensor );

void IRSensor::activity()
{

    int own_addr = gate( "out" )->toGate()->index();
    int track_size = parentModule()->par("IRTrackSize");
    int num_tracks = parentModule()->par("num_Tracks");
    double delay = parentModule()->par("IRTrackDelay");
    cOutVector resp_v("response_time");
    double response_time;
    bool sim_start = true;

    for(;;)
    {
        if (!sim_start)
        {
            // keep an interval between batches of tracks
being sent out
            wait( delay );
        }

        sim_start = false;
        for(int i=0;i<num_tracks; i++)    //send out one
track per object out there.
        {
            // connection setup
            ev << "Client " << name() << " " << own_addr
<< " sending IR Track of size " << track_size << " bits\n";
            cMessage *work = new cMessage( name());

```

```

        work->addPar("src") = own_addr;
        work->addPar("fwd") = true;
        work->setLength(track_size);
        work->setTimestamp();    //puts a current
time timestamp on it.
        response_time = simTime();
        send( work, "out" );
    }
}
}
//-----
// file: SensorInterfaceCapsule.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 15 Nov 2003
// The SensorInterfaceCapsule connects sensors with the SFP
//-----
#include "omnetpp.h"
class SensorInterfaceCapsule : public cSimpleModule
{

Module_Class_Members(SensorInterfaceCapsule,cSimpleModule,1
6384)

    virtual void activity();
};
Define_Module( SensorInterfaceCapsule );
void SensorInterfaceCapsule::activity()
{
    double avg_utilization = 0.0;
    double      process_time      =      parentModule()-
>par("Process_Time");
    cOutVector resp_v("SIC utilization");
    int      num_radarsensors      =      parentModule()-
>par("num_RadarSensors");
    int      num_irsensors      =      parentModule()-
>par("num_IRSensors");

```



```

    int num_sensors = num_radarsensors+num_irsensors;

    for(;;)
    {
        // receive msg (implicit queueing!)
        cMessage *msg = receive();
        // Make sure you put in some delay for handling
of the message
        wait(process_time);
        avg_utilization = avg_utilization + process_time;
        resp_v.record(avg_utilization/simTime());

        // if there is only one or less tracks in the
simulation
        if ( num_sensors < 2 )
        {
            // then send it to the CFC
            ev << "Forwarding msg to CFC" << '\n';
            send( msg, "CFCout");
        }
        else
        { // else there's the possibility that it's a
redundant track, so send it to the TFC
            ev << "Relaying msg to TFC" << '\n';
            send( msg, "TFCout");
        }
    }
}

//-----
// file: CollaborativeFusionCapsule.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 19 Nov 2003
// The Collaborative Fusing Capsule within a Sensor Fusion
Processor

```

```

//-----

#include "omnetpp.h"
class CollaborativeFusionCapsule : public cSimpleModule
{

Module_Class_Members(CollaborativeFusionCapsule,cSimpleModule,16384)

    virtual void activity();
};

Define_Module( CollaborativeFusionCapsule );

void CollaborativeFusionCapsule::activity()
{
    double avg_utilization = 0.0;

    double      process_time      =      parentModule()-
>par("Process_Time");

    int      num_fusion_requests      =      parentModule()-
>par("num_FusionRequests");

    double fusion_time = parentModule()->par("Fusion");

    int      fused_track_size      =      parentModule()-
>par("FusedTrackSize");

    cOutVector resp_v("CFC utilization");

    int num_tracks = parentModule()->par("num_Tracks");

    int      num_radarsensors      =      (parentModule()-
>par("num_RadarSensors"));

    int      num_irsensors      =      (parentModule()-
>par("num_IRSensors"));

    int num_sensors = num_radarsensors+num_irsensors;

    double      fusion_variable      =
(num_tracks/(num_sensors*num_tracks)); //watch out for
divide by 0 errors

    int dropped_tracks = 0;

    int forwarded_tracks = 0;

    int total_tracks = 0; //total tracks received

```

```

for(;;)
{
    cMessage *msg = receive();
    // Make sure you put in some delay for handling
of the message
    wait(process_time);
    avg_utilization = avg_utilization + process_time;
    resp_v.record(avg_utilization/simTime());

    total_tracks++;
    int source = msg->par("src");

    if (msg->arrivedOn("SICin"))
    {
        if (source < num_fusion_requests)
        {
            msg->addPar("CFC") = true;
        }
        else
        {
            msg->addPar("CFC") = false;
        }
        send(msg, "TLCout");
    }

    else if (msg->arrivedOn("TFCin"))
    {
        if (source < num_fusion_requests)
        {
            msg->addPar("CFC") = true;
        }
        else
        {
            msg->addPar("CFC") = false;
        }
    }
}

```

```

    }
    send(msg, "TLCout");
}

else if (msg->arrivedOn("SNICRequestin"))
{
    msg->addPar("CFC") = false;
    //Fuse it
    wait(fusion_time);
    // ensure the size is a fused track size
    msg->setLength(fused_track_size);
    //add a parameter, so it knows this was a
collaboratively fused track
    msg->addPar("ColFus");
    msg->par("ColFus") = true;
    //push it out to SensorNet
    send(msg, "SNICout");
}

bool CFC = msg->par("CFC");
if (msg->arrivedOn("TLCin")&& CFC)
{
    //add a parameter, so it knows this was not
a collaboratively fused track
    msg->addPar("ColFus");
    msg->par("ColFus") = false;

    //request better track from SensorNet
    msg->setLength(50); //THIS IS AN EMBEDDED
PARAMETER!!! (not visible to the NED file)
    send(msg, "SNICRequestout");
}

else if (msg->arrivedOn("TLCin")&& !CFC)
{

```

```

        //add a parameter, so it knows this was not
a collaboratively fused track
        msg->addPar("ColFus");
        msg->par("ColFus") = false;

        send(msg, "SNICout");
    }
}
}

```

```

//-----
// file: SensorNet.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 19 Nov 2003
// The Sensor Net serves and receives work from the SFP.
//-----

```

```

#include "omnetpp.h"

```

```

class SensorNet : public cSimpleModule
{
    Module_Class_Members(SensorNet,cSimpleModule,16384)
    virtual void activity();
};

```

```

Define_Module( SensorNet );

```

```

void SensorNet::activity()
{
    double tracks_received = 0.00000;
    double colfus_tracks_received = 0.00000;
    double          list_delay          =          parentModule()-
>par("TrackListDelay");
    double sim_marker = 0.00000;

```

```

        int        fused_track_size        =        parentModule()-
>par("FusedTrackSize");
        int num_tracks = parentModule()->par("num_Tracks");
        double avg_utilization = 0.00;
        double        process_time        =        parentModule()-
>par("Process_Time");
        cOutVector resp_v("SN.SFPCCommCapsule Utilization");
        double total_StSNT = 0.00000;
        double total_colfus_StSNT = 0.00000;
        double avg_SensortoSensorNetTime = 0.00000;
        double avg_colfus_SensortoSensorNetTime = 0.00000;
        double colfus_SensortoSensorNetTime = 0.00000;
        double SensortoSensorNetTime = 0.00000;
        cOutVector resp_t("Normal Tracks Average Time, Sensors
to SensorNet");
        cOutVector resp_c("ColFus Tracks Average Time, Sensors
to SensorNet");

        for(;;)
        {

                //need to send out the TrackList to all SFPs
periodically, without disrupting everything else
                if (simTime()>sim_marker)
                {

                        cMessage *listmsg = new cMessage( name() );
                        listmsg-
>setLength(num_tracks*fused_track_size);
                        listmsg->addPar("ColFus") = false;
                        scheduleAt(simTime()+list_delay, listmsg);
                        sim_marker = simTime()+list_delay;
                        ev << "Track List prepared for Broadcast" <<
'\n';

                }

                // receive msg (implicit queueing!)

```

```

        cMessage *msg = receive();

        // Make sure you put in some delay for handling
of the message
        wait(process_time);
        avg_utilization = avg_utilization + process_time;
        resp_v.record(avg_utilization/simTime());

        if(msg->isSelfMessage())
        {
            ev << "Size of Track List = " << msg-
>length() << '\n';
            ev << "TRACK LIST BROADCAST" << '\n';
            send(msg, "TrackListout");
        }

        bool ColFus = msg->par("ColFus");

        if(msg->arrivedOn("SFPin")&&!ColFus)
        {
            tracks_received++;
            simtime_t temp = msg->timestamp();
            ev << "Original Timestamp on the message = "
<< temp << '\n';
            simtime_t temp2 = simTime();
            ev << "Timestamp at the SensorNet = " <<
temp2 << '\n';
            SensortoSensorNetTime = temp2 - temp;
            ev << "SensortoSensorNetTime value = " <<
SensortoSensorNetTime << '\n';
            total_StSNT = total_StSNT +
SensortoSensorNetTime;
            avg_SensortoSensorNetTime =
total_StSNT/tracks_received;
            ev << "avg_SensortoSensorNetTime value = "
<< avg_SensortoSensorNetTime << '\n';
            resp_t.record(avg_SensortoSensorNetTime);

```

```

        delete msg;
    }
    else if (msg->arrivedOn("SFPin")&&ColFus)
    {
        colfus_tracks_received++;
        simtime_t tempA = msg->timestamp();
        ev << "COLFUS: Original Timestamp on the
message = " << tempA << '\n';
        simtime_t tempB = simTime();
        ev << "COLFUS: Timestamp at the SensorNet =
" << tempB << '\n';
        colfus_SensortoSensorNetTime    =    tempB    -
tempA;
        ev << "colfus_SensortoSensorNetTime value =
" << colfus_SensortoSensorNetTime << '\n';
        total_colfus_StSNT    =    total_colfus_StSNT    +
colfus_SensortoSensorNetTime;
        avg_colfus_SensortoSensorNetTime    =
total_colfus_StSNT/colfus_tracks_received;
        ev    <<    "avg_colfus_SensortoSensorNetTime
value = " << avg_colfus_SensortoSensorNetTime << '\n';

        resp_c.record(avg_colfus_SensortoSensorNetTime);
        delete msg;
    }

    else if(msg->arrivedOn("SFPRequestin"))
    {
        msg->setLength(fused_track_size);
        //send the message
        send(msg, "SFPRequestout");
    }
}

}

//-----
----

// file: SensorNetInterfaceCapsule.cpp

```



```

// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 19 Nov 2003
// This module connects the SFP to the SensorNet.
//-----
----

#include "omnetpp.h"

class SensorNetInterfaceCapsule : public cSimpleModule
{

Module_Class_Members(SensorNetInterfaceCapsule,cSimpleModule,16384)

    virtual void activity();
};

Define_Module( SensorNetInterfaceCapsule );

void SensorNetInterfaceCapsule::activity()
{
    double avg_utilization = 0.0;

    double          process_time          =          parentModule()-
>par("Process_Time");
    cOutVector resp_v("SNIC Utilization");

    long total_bits = 0;
    double network_util;
    cOutVector resp_n("Network utilization");

    for(;;)
    {
        // receive msg (implicit queueing!)
        cMessage *msg = receive();

        // Make sure you put in some delay for handling
        of the message

```

```

wait(process_time);
avg_utilization = avg_utilization + process_time;
resp_v.record(avg_utilization/simTime());

if (msg->arrivedOn("CFCin"))
{
    send(msg, "SNout");
}
else if (msg->arrivedOn("CFCRequestin"))
{
    send(msg, "SNRequestout");
}
else if (msg->arrivedOn("SNRequestin"))
{
    send(msg, "CFCRequestout");
}
else if (msg->arrivedOn("TrackListin"))
{
    send(msg, "TrackListout");
}
else if (msg->arrivedOn("CFCin"))
{
    send(msg, "SNout");
}

//      if (source == server_add)
//      {
//          total_bits = total_bits + reply_size;
//          network_util  =  total_bits  /  (simTime()  *
data_rates);
//          if (network_util > 1.0) network_util = 1.0;
//          resp_n.record(network_util);
//      }
}

```

```

}
//-----
// file: TrackFusingCapsule.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 15 Nov 2003
// The Track Fusing Capsule within a Sensor Fusion
// Processor
//-----

#include "omnetpp.h"

class TrackFusingCapsule : public cSimpleModule
{

Module_Class_Members(TrackFusingCapsule,cSimpleModule,16384
)

    virtual void activity();
};

Define_Module( TrackFusingCapsule );

void TrackFusingCapsule::activity()
{
    double avg_utilization = 0.0;

    double      process_time      =      parentModule()-
>par("Process_Time");

    double fusion_time = parentModule()->par("Fusion");

    int      fused_track_size      =      parentModule()-
>par("FusedTrackSize");

    cOutVector resp_v("TFC utilization");

    double num_tracks = parentModule()->par("num_Tracks");

    int      num_radarsensors      =      (parentModule()-
>par("num_RadarSensors"));

    int      num_irsensors      =      (parentModule()-
>par("num_IRSensors"));

```

```

double num_sensors = num_radarsensors+num_irsensors;
double fusion_variable = 0.000;
double random_num;
int dropped_tracks = 0;
int forwarded_tracks = 0;
int total_tracks = 0; //total tracks received
int own_addr = gate( "CFCout" )->toGate()->index();

for(;;)
{
    cMessage *msg = receive();
    // Make sure you put in some delay for handling
of the message
    wait(process_time);
    avg_utilization = avg_utilization + process_time;
    resp_v.record(avg_utilization/simTime());

    fusion_variable = (1/num_sensors); //watch out
for divide by 0 errors

    if(total_tracks<num_tracks) //this lets the first
track through for each actual object out there.
    {
        fusion_variable = 1.000;
    }

    if (msg->arrivedOn("SICin"))
    {
        ev << "TFC --> TLC" << '\n';
        send(msg, "TLCout");
    }
    else if (msg->arrivedOn("TLCin"))
    {
        total_tracks++;
    }
}

```

```

        random_num = uniform(.01,1); //we're going
to drop all but fusion variable % of messages.

        //let's see what the variables
are!*****

        ev << "TrackFusingCapsule, deciding if we
should fuse or correlate" <<'\n';

        ev << "Here's the Random Number --> " <<
random_num <<'\n';

        ev << "Here's the Fusion Variable --> " <<
fusion_variable << '\n';

        //If the Random Number is larger, it's
correlated (dropped)

        //Elsewise, it's fused with the correlated
tracks and forwarded

        if (random_num > fusion_variable) //need to
drop the message and wait for the next one.
        {
            ev << "@@TFC Dropping the message!@"
<< '\n';

            delete msg;
            dropped_tracks++;
        }
        else //fuse, then forward the track to the
Collaborative Fusion Capsule
        {

            ev << "@@TFC Fusing the message!@" <<
'\n';

            //Note, the actual fusion request is
passed to an internal capsule, clearing the
            //TFC to handle other incoming tracks.
This is modeled by subtracting fusion_time
            //from the fused message's timestamp.
This shows the time delay in the end.

            simtime_t temp = msg->timestamp();

```

```

        if (temp>fusion_time)
        {
            temp = temp - fusion_time;
        }
        msg->setTimestamp(temp);

        // change the size to a fused track
size      msg->setLength(fused_track_size);

        ev << "TFC --> CFC" << '\n';
        // forward the track
        send(msg, "CFCout");
        forwarded_tracks++;
    }
}
else
{
    ev << "***ERROR: TFC did not handle
message!***";
}
}

//-----
// file: TrackListCapsule.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 14 Nov 2003
// The Track List Capsule keeps track of the track list
// for the SFP.
//-----

#include "omnetpp.h"

class TrackListCapsule : public cSimpleModule

```

```

{

Module_Class_Members(TrackListCapsule,cSimpleModule,16384)
    virtual void activity();
};

Define_Module( TrackListCapsule );

void TrackListCapsule::activity()
{
    double avg_utilization = 0.0;
    double      process_time      =      parentModule()-
>par("Process_Time");
    double check_time = parentModule()->par("ListCheck");
    cOutVector resp_v("TLC Utilization");
    int num_tracks = parentModule()->par("num_Tracks");

    for(;;)
    {

        cMessage *msg = receive();
        // Make sure you put in some delay for handling
of the message
        wait(process_time);
        avg_utilization = avg_utilization + process_time;
        resp_v.record(avg_utilization/simTime());

        if (msg->arrivedOn("TFCin"))
        {
            ev << "TrackListCapsule processing msg from
TFCin" << '\n';

            //Put in check time for checking the list
            wait(check_time);
            avg_utilization      =      avg_utilization      +
check_time;

```

```

        resp_v.record(avg_utilization/simTime());

        send(msg, "TFCout");
    }
    else if (msg->arrivedOn("CFCin"))
    {
        ev << "TrackListCapsule processing msg from
CFCin" << '\n';
        wait(check_time);

        //Put in check time for checking the list
        wait(check_time);
        avg_utilization      =      avg_utilization      +
check_time;

        resp_v.record(avg_utilization/simTime());

        send(msg, "CFCout");
    }
    else if (msg->arrivedOn("TrackListin"))
    {
        ev << "TrackListCapsule processing msg from
TrackListin" << '\n';
        //note, this causes minimal delay, as it
        goes to the inactive Track Registry
        //then, after the inactive comes on line, it
        is given to the formerly active TR.

        //This is an estimated service delay to
        switch between the active and semi-active TRs.
        wait(process_time);
        avg_utilization      =      avg_utilization      +
process_time;

        resp_v.record(avg_utilization/simTime());

        delete msg;
    }

```



```

    }
}

```

B. SENSOR NET SIMULATION.

This simulation was a follow on to our SFP Simulation work. It is complete and functional except for the BMC2 generating the firing order correctly and sending it through the PHIC → TCC → TRC → TSC Chain and the WP → TSC → SFPIC Chain. This can well serve as a basis for analysis of different architectural concepts for the Sensor Net.

```

//-----
----

// file: SensorNetSim.ned
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 25 Nov 2003
//-----
----

// RadarSensor --
//
// A ground based radar sensor which sends cues to the
// SensorNet.
//
simple RadarSensor
    gates:
        out: out;
endsimple

// IRSensor --
//

```

```

// A satellite based IR sensor which sends cues to the
SensorNet.
//
simple IRSensor
    gates:
        out: out;
endsimple

// Sensor Controlling Authority --
//
// The entity which issues orientation commands to the
individual sensors
//
simple SCA
    gates:
        in: in; //this is where the SCA receives the master
track list
endsimple

// Competent Authority --
//
// Any authority which can competently give cues to
SensorNet
//
simple CA
    gates:
        out: out;
endsimple

// Remote SensorNet --
//
// A peer sensornet to our sensornet.

```

```

//
simple RemoteSN
    gates:
        in: in; //receives the master track list
        (abstracted version) from our SensorNet
        out: out; //pushes its abstracted master track list
        to our SensorNet
        out: Cueout; //pushes cues to the cueing capsule of
        our SensorNet
endsimple

```

```

// SensorFusionProcessor --
//
// Responsible for pushing tracks to the Sensor Net.
// Requests tracks for
// Collaborative Fusion from SensorNet.
//
simple SFP
    gates:
        out: SNRequestout[]; //port used to request
        collaboratively fused tracks from SN
        in: SNRequestin[]; //port used to receive
        collaboratively fused tracks from SN
        in: TrackListin; //port used to receive the master
        track list from SN
        out: SNout[]; //port used to push tracks to SN
endsimple

```

```

// SFP Interface Capsule --
//
// Pushes master track list to the SFP, receives tracks
// from the SFP, and handles
// requests for tracks from other SFPs.
//

```

```

simple SFPIC
    gates:
        out:  SFPRequestout[];  //port  used  to  push
collaboratively fused tracks to the SFP
        in: SFPRequestin[]; //port used to receive c. fused
track requests from SFP
        in: TrackListin; //port used to receive the master
track list from the TSC
        out: TrackListout[]; //port used to push the master
track list to the SFPs
        in: SFPin[]; //port used to receive tracks from
SFPs, # = # of SFPs
        out: TCCout[]; //port used to push tracks to the
TCC, # = # of SFPs
        out: WPout[]; //port used to push tracks to the
WPIC
        in: Shortin; //port used to receive short orders
from the TSC
endsimple

```

```

// Track Correlation Capsule
//
//
//
simple TCC
    gates:
        in: TRCin; //receives the master track list from
the TRC
        out: TRCout[]; //pushes modifications (tracks) to
the TRC, # of ports = # of tracks
        in: SFPICin[]; //receives tracks from the SFP
Interface Capsule
        in: PHICBMC2in[]; //receives modifications to
tracks
        in: PHICPSNin[]; //receives abstracted master track
lists
        in: Cuein[]; //# = # of tracks

```

```

        out: Cueout[]; //# = # of tracks
endsimple

// Cueing Capsule
//
//
//
simple CC
    gates:
        in: Cuein[]; //# of ports = Sensors + Competent
        Authorities + Peer SensorNets
        out: TCCout[]; //# of ports = # of tracks
        in: TCCin[]; //# of ports = # of tracks
        out: SCAout[]; //# of ports = # of SCAs
endsimple

// Track Registry Capsule
//
//
//
simple TRC
    gates:
        in: TCCin[]; //receives modifications (tracks) from
        the TCC, # of ports = # of tracks
        out: TCCout; //pushes master track list to TCC
        out: TSCout; //pushes master track list to TSC
endsimple

// Track Server Capsule
//
//
//

```

```

simple TSC
  gates:
    in: TRCin; //receives master track list from TRC
    in: WPICin[]; //receives requests from WPICs for
tracks, # = # of WPs
    out: Shortout; //passes short commands to the SFPIC
    out: TrackListout; //passes the master track list
to the WPIC
    out: BMC2out; //passes the master track list to the
PHIC
    out: PSNout; //passes the abstracted track list to
the PHIC
endsimple

```

```

// Peer/Higher Interface Capsule

```

```

//

```

```

//

```

```

//

```

```

simple PHIC

```

```

  gates:
    in: MasterListin;
    in: AbstractListin;
    out: TCCBMC2out[]; //used to pass BMC2
modifications to the TCC
    out: TCCPSNout[]; //used to pass abstracted master
track lists from PSNs
    in: BMC2in[]; //used to receive BMC2 modifications,
# of ports = # of tracks
    out: BMC2out; //used to pass the Master Track List
to the BMC2
    out: PSNout[]; //used to pass abstracted master
track lists to Peer SensorNets
    in: PSNin[]; //used to receive abstracted master
track lists from Peer SensorNets
endsimple

```

```

// BMC2 --
//
// The battle management (what the army/navy/marines call
command and control) element.
//
simple BMC2
    gates:
        in: in; //receives master track list from the
SensorNet
        out: out[]; //pushes its modifications back to the
SensorNet, # of ports = # of tracks
        out: WPout[]; //pushes firing orders to the Weapons
Platforms
endsimple

```

```

// Weapon Platform
//
// A weapon's controlling entity. It requests firing
solution quality tracks from the SensorNet
// so that it may fire on tasked targets.
//
simple WP
    gates:
        in: in; //receives the firing solution quality
track from the SensorNet
        out: out; //requests the firing solution quality
track from the SensorNet
        in: BMC2in; //receives firing orders from the BMC2
endsimple

```

```

// Weapon Platform Interface Capsule
//

```

```

//
//
simple WPIC
    gates:
        out: WPout[]; //port used to pass tracks to WPs
        in: WPin[]; //port used to receive requests from
the WPs
        out: TrackRequestout[]; //port used to request a
track (to TSC), # of ports = # of requests
        in: TrackRequestin[]; //port used to receive
requested tracks (from SFPIC)
endsimple

// SensorNetSim --
//
// Model of the SensorNet, with connections to all
interacting devices
//
module SensorNetSim
    parameters:
        //parameters that involve only one entity
        data_rate_RadarSensorToSFP : numeric, // the data
rate between Radar Sensor and the SFP
        RadarTrackSize : numeric, // size of an unfused
radar track
        data_rate_IRSensorToSFP : numeric, // the data rate
between IR Sensor and the SFP
        IRTrackSize : numeric, // size of an unfused IR
track
        data_rate_SFPTtoSensorNet : numeric, // the data
rate between the SFP and SensorNet
        ClassDelay : numeric, // time required to classify
a track as either target or not
        TrackDelay : numeric, // time required to get a
track to SensorNet (from SFPsim).

```



```

        //parameters that involve more than one entity
        num_RadarSensors : numeric, // the number of Radar
Sensors
        num_IRSensors : numeric, // the number of IR
Sensors
        num_SFPS : numeric, // the number of SFPS in the
simulation
        num_SCAs : numeric, // the number of Sensor
Controlling Authorities
        num_CAs : numeric, // the number of Competent
Authorities
        num_PSNs : numeric, // the number of Peer
SensorNets
        num_WPs : numeric, // the number of Weapon
Platforms
        num_Tracks : numeric, // the number of actual
tracks out there (ie: planes, missiles, etc)
        num_Targets : numeric, // the number of targets out
there (ie: enemy rockets, missiles, etc)
        num_FusionRequests : numeric, // maximum number of
collaborative fusion requests per SFP
        FusedTrackSize : numeric, // size of a firing
solution quality fused track
        Process_Time : numeric, // Generic handling time
each module eats in handling a track
        ListCheck : numeric, // Time Required to Check a
Track against the List
        Fusion : numeric, //Time Required to perform a
Fusing Action

```

```

        //parameters for the SensorNet
        data_rate_Internal : numeric, // data rate of
connections within the SFP
        TrackListDelay : numeric; //amount of delay between
sendings of the master track list

```

submodules:

```

        RadarSensor: RadarSensor[num_RadarSensors];

```

```

        display:
"p=361,492,r,100;i=radarsensoricon;b=92,88";
        IRSensor: IRSensor[num_IRSensors];
        display:
"p=371,373,r,90;i=satellitesensoricon;b=75,97";
        SCA: SCA[num_SCAs];
        display: "p=363,285,r,90;i=telnet;b=38,28";
        CA: CA[num_CAs];
        display: "p=51,477,r,90;i=telnet;b=38,28";
        RemoteSN: RemoteSN[num_PSNs];
        display: "p=355,225,r,90;i=router;b=32,32";
        SFP: SFP[num_SFPs];
        gatesizes:
            SNRequestout[num_Tracks], //port used to
request collaboratively fused tracks from SN
            SNRequestin[num_Tracks], //port used to
receive collaboratively fused tracks from SN
            SNout[num_Tracks]; //port used to push
tracks to SN
            display: "p=51,25,r,90;i=cogwheel;b=32,30";
        SFPIC: SFPIC;
        gatesizes:
            SFPRequestout[num_Tracks*num_SFPs], //port
used to push collaboratively fused tracks to the SFP
            SFPRequestin[num_Tracks*num_SFPs], //port
used to receive c. fused track requests from SFP
            TrackListout[num_SFPs], //port used to push
the master track list to the SFPs
            SFPin[num_SFPs*num_Tracks], //port used to
receive tracks from SFPs
            TCCout[num_SFPs*num_Tracks], //port used to
push tracks to the TCC, # = # of SFPs
            WPout[num_Targets]; //port used to push
tracks to the WPIC
            display: "p=119,195,r,70;i=comp;b=36,32";
        TCC: TCC;
        gatesizes:

```

```

        TRCout[num_Tracks], //pushes modifications
(tracks) to the TRC, # of ports = # of tracks
        SFPICin[num_Tracks * num_SFPs], //receives
tracks from the SFP Interface Capsule
        PHICBMC2in[num_Targets],           //receives
modifications to tracks
        PHICPSNin[num_PSNs], //receives abstracted
master track lists
        Cuein[num_Tracks], //# = # of tracks
        Cueout[num_Tracks]; //# = # of tracks
        display: "p=79,311,r,70;i=comp;b=36,32";
CC: CC;
        gatesizes:
            Cuein[num_RadarSensors + num_IRSensors +
num_CAs + num_PSNs],
            TCCout[num_Tracks], //# of ports = # of
tracks
            TCCin[num_Tracks], //# of ports = # of
tracks
            SCAout[num_SCAs]; //# of ports = # of SCAs
        display: "p=227,407,r,70;i=comp;b=36,32";
TRC: TRC;
        gatesizes:
            TCCin[num_Tracks]; //receives modifications
(tracks) from the TCC, # of ports = # of tracks
        display: "p=163,259,r,70;i=comp;b=36,32";
TSC: TSC;
        gatesizes:
            WPICin[num_Targets]; //receives requests
from WPICs for tracks, # = # of Targets
        display: "p=247,259,r,70;i=comp;b=36,32";
PHIC: PHIC;
        gatesizes:
            TCCBMC2out[num_Targets], //used to pass
BMC2 modifications to the TCC
            TCCPSNout[num_PSNs], //used to pass
abstracted master track lists from PSNs

```

```

        BMC2in[num_Targets], //used to receive BMC2
modifications, # of ports = # of tracks

        PSNout[num_PSNs], //used to pass abstracted
master track lists to Peer SensorNets

        PSNin[num_PSNs]; //used to receive
abstracted master track lists from Peer SensorNets

        display: "p=235,347,r,70;i=comp;b=36,32";
BMC2: BMC2;

        gatesizes:

                out[num_Targets], //pushes its
modifications back to the SensorNet, # of ports = # of
targets

                WPout[num_WPs]; //pushes firing orders to
WPs

                display: "p=331,173,r,90;i=telnet;b=38,28";
WP: WP[num_WPs];

        display:
        "p=219,97,r,90;i=weaponplatformicon;b=39,73";
WPIC: WPIC;

        gatesizes:

                WPout[num_WPs], //port used to pass tracks
to WPs

                WPin[num_WPs], //port used to receive
requests from the WPs

                TrackRequestout[num_Targets], //port used
to request a track (to TSC), # of ports = # of requests

                TrackRequestin[num_Targets]; //port used to
receive requested tracks (from SFPIC)

                display: "p=255,183,r,70;i=comp;b=36,32";

        //see p43 of the manual for figuring out problems with
ports (especially the [] parts)

        connections:

                //connect up the RadarSensors to the CC

        for i=0..num_RadarSensors-1 do
                RadarSensor[i].out --> delay 5ms -->
CC.Cuein[i];
        endfor;

```

```

//connect up the IR Sensors to the CC also
for i=0..num_IRSensors-1 do
    IRSensor[i].out    -->    delay    500ms    -->
CC.Cuein[num_RadarSensors + i];
endfor;

//connect up the SCAs to the CC
for i=0..num_SCAs-1 do
    SCA[i].in <-- delay 5ms <-- CC.SCAout[i];
endfor;

//connect up the CAs to the CC
for i=0..num_CAs-1 do
    CA[i].out    -->    delay    5ms    -->
CC.Cuein[num_RadarSensors + num_IRSensors + i];
endfor;

//connect up the Remote SensorNets
for i=0..num_PSNs-1 do
    RemoteSN[i].in    <--    delay    50ms    <--
PHIC.PSNout[i];
endfor;
for i=0..num_PSNs-1 do
    RemoteSN[i].out    -->    delay    50ms    -->
PHIC.PSNin[i];
endfor;

for i=0..num_PSNs-1 do
    RemoteSN[i].Cueout    -->    delay    50ms    -->
CC.Cuein[num_RadarSensors + num_IRSensors + num_CAs + i];
endfor;

//connect up the BMC2
BMC2.in <-- delay 5ms <-- PHIC.BMC2out;
for i=0..num_Targets-1 do

```

```

        BMC2.out[i] --> delay 5ms --> PHIC.BMC2in[i];
    endfor;
    for i=0..num_WPs-1 do
        BMC2.WPout[i] --> delay 5ms --> WP[i].BMC2in;
    endfor;

    //connect up the Weapons Platforms
    for i=0..num_WPs-1 do
        WP[i].out --> delay 5ms --> WPIC.WPin[i];
    endfor;
    for i=0..num_WPs-1 do
        WP[i].in <-- delay 5ms <-- WPIC.WPout[i];
    endfor;

    //connect up the SFPs
    for i=0..num_SFPs-1, j=0..num_Tracks-1 do
        SFP[i].SNRequestout[j] --> delay 5ms -->
SFPIC.SFPRequestin[(i*num_Tracks)+j];
    endfor;
    for i=0..num_SFPs-1, j=0..num_Tracks-1 do
        SFP[i].SNRequestin[j] <-- delay 5ms <--
SFPIC.SFPRequestout[(i*num_Tracks)+j];
    endfor;
    for i=0..num_SFPs-1 do
        SFP[i].TrackListin <-- delay 5ms <--
SFPIC.TrackListout[i];
    endfor;
    for i=0..num_SFPs-1, j=0..num_Tracks-1 do
        SFP[i].SNout[j] --> delay 5ms -->
SFPIC.SFPin[(i*num_Tracks)+j];
    endfor;

    //connect up the SFPIC
    SFPIC.TrackListin <-- delay 0ms <--
TSC.TrackListout;
    SFPIC.Shortin <-- delay 0ms <-- TSC.Shortout;

```

```

        for i=0..(num_SFps*num_Tracks)-1 do
            SFPIC.TCCout[i]    -->    delay    0ms    -->
TCC.SFPICin[i];
        endfor;
        for i=0..num_Targets-1 do
            SFPIC.WPout[i]    -->    delay    0ms    -->
WPIC.TrackRequestin[i];
        endfor;

        //connect up the WPIC
        for i=0..num_Targets-1 do
            WPIC.TrackRequestout[i]    -->    delay    0ms    -->
TSC.WPICin[i];
        endfor;

        //connect up the PHIC
        PHIC.MasterListin <-- delay 0ms <-- TSC.BMC2out;
        PHIC.AbstractListin <-- delay 0ms <-- TSC.PSNout;
        for i=0..num_Targets-1 do
            PHIC.TCCBMC2out[i]    -->    delay    0ms    -->
TCC.PHICBMC2in[i];
        endfor;
        for i=0..num_PSNs-1 do
            PHIC.TCCPSNout[i]    -->    delay    0ms    -->
TCC.PHICPSNin[i];
        endfor;

        //connect up the CC
        for i=0..num_Tracks-1 do
            CC.TCCout[i] --> delay 0ms --> TCC.Cuein[i];
        endfor;
        for i=0..num_Tracks-1 do
            CC.TCCin[i] <-- delay 0ms <-- TCC.Cueout[i];
        endfor;

        //connect up the TCC

```

```

TCC.TRCin <-- delay 0ms <-- TRC.TCCout;
for i=0..num_Tracks-1 do
    TCC.TRCout[i] --> delay 0ms --> TRC.TCCin[i];
endfor;

//connect up the TRC
TRC.TSCout --> delay 0ms --> TSC.TRCin;

display: "p=34,162;b=249,277";
endmodule

//
// Instantiates a SensorNet
//
network TheSensorNetSim : SensorNetSim // must match file
name (e.g. test.ned)
    parameters:
        data_rate_RadarSensorToSFP = input(1440000, "Data
rate (bps) between Radar Sensors and the SFP:_____"),
        RadarTrackSize = input(1024, "Size
(bits) of an unfused radar track:_____"),
        data_rate_IRSensorToSFP = input(93000, "Data
rate (bps) between IR Sensor and the SFP:_____"),
        IRTrackSize = input(256, "Size
(bits) of an unfused IR track:_____"),
        data_rate_SFPtoSensorNet = input(45000000, "The
data rate (bps) between the SFP and SensorNet:_____"),
        ClassDelay = input(.005,
"Classification Delay (sec) to decide target/not
target:_____"),
        TrackDelay = input(.2,
"SFP Time Delay (sec) for Sensor to
SensorNet:_____"),
        num_RadarSensors = input(1, "Number
of Ground-Based Radar Sensors:_____"),

```



```

        num_IRSensors = input(1,
        "Number of Satellite-Based IR
Sensors:_____"),
        num_SFPPs = input(1, "Number
of SFPPs in the simulation:_____"),
        num_SCAs = input(1, "Number
of Sensor Controlling Authorities:_____"),
        num_CAs = input(1, "Number
of Competent Authorities:_____"),
        num_PSNs = input(1, "Number
of Peer SensorNets:_____"),
        num_WPs = input(1, "Number
of Weapon Platforms:_____"),
        num_Tracks = input(2, "Number
of actual Tracks (ie: planes, missiles, etc):_____"),
        num_Targets = input(2, "Number
of actual Targets (ie: enemy rockets, missiles):____"),
        num_FusionRequests = input(1,
        "Maximum number of ColFus requests per SFP:_____"),
        FusedTrackSize = input(1152, "Size
of a firing solution quality fused track:_____"),
        Process_Time = input(.000005, "Time
(sec) each Module takes to handle a track/target:_____"),
        ListCheck = input(.0005, "Time
(sec) to check a track against the List:_____"),
        Fusion = input(.01,
        "Time (sec) required to perform a Fusing
Action:_____"),
        data_rate_Internal = input(1000000000, "Data
Rate (bps) between Capsules:_____"),
        TrackListDelay = input(.1, "Delay
(sec) between Master Track List broadcasts:_____");
endnetwork
//-----
----
// file: BMC2.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 29 Nov 2003

```

```

// The BMC2 controls which tracks are targets and which are
not
//-----
----

#include "omnetpp.h"

class BMC2 : public cSimpleModule
{
    Module_Class_Members(BMC2,cSimpleModule,16384)
    virtual void activity();
};

Define_Module( BMC2 );

void BMC2::activity()
{
    int num_tracks = parentModule()->par("num_Tracks");
    int num_targets = parentModule()->par("num_Targets");
    int      num_radarsensors      =      parentModule()-
>par("num_RadarSensors");
    int      num_irsensors         =      parentModule()-
>par("num_IRSensors");
    int num_sensors = num_radarsensors+num_irsensors;
    int num_wps = parentModule()->par("num_WPs");
    double avg_utilization = 0.0;
    double      process_time      =      parentModule()-
>par("Process_Time");
    double      classification_delay      =      parentModule()-
>par("ClassDelay");
    cOutVector resp_v("BMC2 utilization");

    int target_list[100];

    for (int t=0; t<100;t++)
    {

```

```

        target_list[t] = -1;
    }

    for(;;)
    {
        // receive msg (implicit queueing!)
        cMessage *msg = receive();
        // Make sure you put in some delay for
handling of the message
        wait(process_time);
        avg_utilization      =      avg_utilization      +
process_time;
        resp_v.record(avg_utilization/simTime());

        if(msg->hasPar("tlp"))
        {

            int      *target      =      (int      *)      msg-
>getObject("TargetList");
            //This gives us the target array that
we will now break down.
            for (int i=0; i<num_targets; i++)
            {
                if (target_list[i] == -1)
                {
                    //Since all this is done at
the beginning of the simulation, it's
                    //impossible to model it
offline by decrementing a timestamp, since that
                    //would put it in the
negatives (an invalid value).
                    //For purposes of this
simulation, we're classifying in a sequential fashion.
                    wait(classification_delay);
//here's the waiting time to classify.
                    int tk = uniform(0,num_wps);
//here we decide which weapon platform should get it.

```

```

        target_list[i] = tk;
        ev << "Target " << i << "
assigned to WP " << target_list[i] << '\n';
        cMessage *fire_order = new
cMessage( name());

        //here's the weapon platform
that was assigned to the target
        fire_order->addPar("BMC2_wp")
= target_list[i];

        //here's the target the
weapon platform is assigned to
        fire_order->addPar("target")
= i; //tracks & targets are counted 0 to n-1.
        cMessage *copy = (cMessage *)
fire_order->dup();

        send(copy, "out"); //we're
sending a copy to the Target List

        send(fire_order, "WPout",
target_list[i]); //simultaneously we send the fire order to
the WP.

    }
    else
    {
        ev << "Target " << i << "
already assigned to WP " << target_list[i] << '\n';
    }
}

}
else
{
    ev << "ERROR: BMC2 received a target
list message with no list attached!!!" << '\n';
}

}

}
//-----
----
```

```

// file: CA.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 3 Dec 2003
// This simulates the abstract concept that others can pass
// cues to the SensorNet (not just sensors and peerSNs).
//-----
----

#include "omnetpp.h"

class CA : public cSimpleModule
{
    Module_Class_Members(CA,cSimpleModule,16384)
    virtual void activity();
};

Define_Module( CA );

void CA::activity()
{
    int own_addr = gate( "out" )->toGate()->index();
    int track_size = parentModule()->par("IRTrackSize");
    int num_tracks = parentModule()->par("num_Tracks");

    for(int i=0;i<num_tracks;i++)
    {
        // connection setup
        ev << "Client " << name() << " " << own_addr << "
sending Cue of size " << track_size << " bits\n";
        cMessage *work = new cMessage( name());
        work->setLength(track_size);
        work->addPar("src") = own_addr;
        work->addPar("track") = i;
    }
}

```

```

        work->setTimestamp();    //puts a current time
timestamp on it.

        send( work, "out" );

        ev << "In CA Module at point 1" << '\n';
        /*******

    }
}
//-----
----
// file: CC.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 29 Nov 2003
// The Cueing Capsule within a SensorNet
//-----
----

#include "omnetpp.h"

class CC : public cSimpleModule
{
    Module_Class_Members(CC,cSimpleModule,16384)
    virtual void activity();
};

Define_Module( CC );

void CC::activity()
{
    double avg_utilization = 0.0;
    double      process_time      =      parentModule()-
>par("Process_Time");
    cOutVector resp_v("CFC utilization");

```

```

    double num_tracks = parentModule()->par("num_Tracks");
    double      num_radarsensors      =      (parentModule()-
>par("num_RadarSensors"));
    double      num_irsensors         =      (parentModule()-
>par("num_IRSensors"));
    double num_sensors = num_radarsensors+num_irsensors;
    double num_cas = parentModule()->par("num_CAs");
    double num_psns = parentModule()->par("num_PSNs");
    double      cue_variable          =
(num_tracks/((num_sensors+num_cas+num_psns)*num_tracks));
//watch out for divide by 0 errors
    int num_scas = parentModule()->par("num_SCAs");

    ev << "In CC Module at point 1" << '\n';
    //*****

    //The cued array will be used to see if we keep or
drop a cue. Note, only cues relevant to our tracks
    //are being received. No irrelevant cues are being
sent by peer SensorNets.
    bool cued[100];
    for (int t=0; t<100;t++)
    {
        cued[t] = false;
    }

    for(;;)
    {
        cMessage *msg = receive();
        // Make sure you put in some delay for handling
of the message
        wait(process_time);
        avg_utilization = avg_utilization + process_time;
        resp_v.record(avg_utilization/simTime());

        int track = msg->par("track");

```

```

int source = msg->par("src");

if (msg->arrivedOn("Cuein"))
{
    if(cued[track] == false)
    {
        ev << "CC: Cue received for track " <<
track << " from " << source << '\n';
        ev << "Reminder: sources are numbered 0
to num_RadarSensors + num_IRSensors+ num_CAs + num_PSNs"
        << '\n';

        double random_num = uniform(.01,1);
//we're going to drop all but fusion variable % of Cues.

//let's see what the variables
are!*****
        ev << "CueingCapsule, seeing if the cue
is redundant" <<'\n';
        ev << "Here's the Random Number --> "
<< random_num <<'\n';
        ev << "Here's the Cue Variable --> " <<
cue_variable << '\n';
        //If the Random Number is larger, it's
forwarded to the TCC
        //Elsewise, it's considered a definite
valid cue and handled as such

        if (random_num > cue_variable) //need
to forward the track to the TCC for a decision
        {
            ev << "Forwarding the cue to the
TCC for a decision" << '\n';
            send(msg, "TCCout");
        }
        else

```



```

        {
            ev << "TRACK ALERT:    Valid cue
received on Track " << track << '\n';
            cued[track] = true;
            for(int s=0; s<num_scas-1; s++)
            {
                cMessage *copy = (cMessage *)
msg->dup();

                send(copy, "SCAout", s);
            }
            send(msg, "SCAout", num_scas-1);
        }
    }
    else
    {
        ev << "CC: Redundant Cue for Target "
<< track << " dropped" << '\n';
        delete msg;
    }
}
else if (msg->arrivedOn("TCCin"))
{
    //if it comes back from the TCC, then it is
a valid cue.  Elsewise it would have been dropped.
    ev << "TRACK ALERT:    Valid cue received for
Track " << track << '\n';
    cued[track] = true;
    for(int s=0; s<num_scas-1; s++)
    {
        cMessage *copy = (cMessage *) msg-
>dup();

        send(copy, "SCAout", s);
    }
    send(msg, "SCAout", num_scas-1);
}
}

```

```

}
//-----
----
// file: IRSensor.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 22 Nov 2003
// This is a generic IR sensor.
//-----
----

#include "omnetpp.h"

class IRSensor : public cSimpleModule
{
    Module_Class_Members(IRSensor,cSimpleModule,16384)
    virtual void activity();
};

Define_Module( IRSensor );

void IRSensor::activity()
{
    int own_addr = gate( "out" )->toGate()->index();
    int track_size = parentModule()->par("IRTrackSize");
    int num_tracks = parentModule()->par("num_Tracks");

    for(int i=0;i<num_tracks;i++)
    {
        // connection setup
        ev << "Client " << name() << " " << own_addr << "
sending IR Cue of size " << track_size << " bits\n";
        cMessage *work = new cMessage( name());
        work->setLength(track_size);
        work->addPar("src") = own_addr;
    }
}

```

```

        work->addPar("track") = i;
        work->setTimestamp();    //puts a current time
timestamp on it.
        send( work, "out" );
    }
}

//-----
----
// file: PHIC.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 3 Dec 2003
// The Peer/Higher Interface Capsule within the SensorNet
//-----
----

#include "omnetpp.h"

class PHIC : public cSimpleModule
{
    Module_Class_Members(PHIC,cSimpleModule,16384)
    virtual void activity();
};

Define_Module( PHIC );

void PHIC::activity()
{
    double avg_utilization = 0.0;
    double      process_time      =      parentModule()-
>par("Process_Time");
    int num_psnns = parentModule()->par("num_PSNs");
    cOutVector resp_v("PHIC utilization");

```

```

    ev << "In PHIC Module at point 1" << '\n';
    /*******

for(;;)
{
//      if (simTime()==0)
///      {
//          ev << "PHIC at the beginning of simulation"
<< '\n';
//          cMessage *work = new cMessage( name());
//          send(work, "BMC2out");
//      }

    ev << "In PHIC Module before receive()" << '\n';
    /*******

    cMessage *msg = receive();

    ev << "In PHIC Module after receive()" << '\n';
    /*******

    // Make sure you put in some delay for handling
of the message
    wait(process_time);
    avg_utilization = avg_utilization + process_time;
    resp_v.record(avg_utilization/simTime());

    if (msg->arrivedOn("MasterListin"))
    {
        ev << "PHIC sending Master Track List to
BMC2" << '\n';
        send(msg, "BMC2out");
    }

```

```

        else if (msg->arrivedOn("AbstractListin"))
        {
            ev << "PHIC sending Abstracted Track List to
PSNs" << '\n';
            for(int s=0; s<num_psns-1; s++)
            {
                cMessage *copy = (cMessage *) msg-
>dup();
                send(copy, "PSNout", s);
            }
            delete msg;
            // send(msg, "PSNout", num_psns);
        }
        else if (msg->arrivedOn("PSNin"))
        {
            //Pass abstracted track lists to the TCC for
inclusion into the master track list.
            //These are not passed for cueing
purposes!!!
            ev << "PHIC sending Abstracted Track List to
TCC" << '\n';
            send(msg, "TCCPSNout");
        }
        else if (msg->arrivedOn("BMC2in"))
        {
            ev << "PHIC sending Master Track List to
TCC" << '\n';
            send(msg, "TCCBMC2out");
        }
        else
        {
            ev << "PHIC inactive" << '\n';
        }
    }
}

```

```

//-----
----
// file: RadarSensor.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 22 Nov 2003
// This is a generic radar sensor.
//-----
----

#include "omnetpp.h"

class RadarSensor : public cSimpleModule
{
    Module_Class_Members(RadarSensor,cSimpleModule,16384)
    virtual void activity();
};

Define_Module( RadarSensor );

void RadarSensor::activity()
{
    int own_addr = gate( "out" )->toGate()->index();
    int track_size = parentModule()->par("RadarTrackSize");
    int num_tracks = parentModule()->par("num_Tracks");

    for(int i=0;i<num_tracks;i++)
    {
        // connection setup
        ev << "Client " << name() << " " << own_addr << "
sending Radar Cue of size " << track_size << " bits\n";
        cMessage *work = new cMessage( name());
        work->setLength(track_size);
        work->addPar("src") = own_addr;
        work->addPar("track") = i;
    }
}

```

```

        work->setTimestamp();    //puts a current time
timestamp on it.
        send( work, "out" );
    }
}

//-----
----
// file: RemoteSN.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 3 Dec 2003
// A peer SensorNet to our SensorNet.
//-----
----

#include "omnetpp.h"

class RemoteSN : public cSimpleModule
{
    Module_Class_Members(RemoteSN,cSimpleModule,16384)
    virtual void activity();
};

Define_Module( RemoteSN );

void RemoteSN::activity()
{
    ev << "In RemoteSN Module at point 1" << '\n';
    //*****

    int own_addr = gate( "out" )->toGate()->index();
    int track_size = parentModule()->par("IRTrackSize");
    int num_tracks = parentModule()->par("num_Tracks");
    double avg_utilization = 0.0;

```

```

        double        process_time        =        parentModule()-
>par("Process_Time");
        double        classification_delay    =        parentModule()-
>par("ClassDelay");
        cOutVector resp_v("BMC2 utilization");

        for(int i=0;i<num_tracks;i++)
        {
                double random_num = uniform(.01,1); //we're only
going to generate 50% of Cues.
                if (random_num > .5) //The cue we were going to
send to our peer sensor net does not
                                //apply to them, so we need to
wait until we have one that does apply.
                {
                        ev << "##TCC Dropping the Cue!##" << '\n';
                }
                else //validate the cue as being relevant to our
SensorNet
                {
                        // connection setup
                        ev << "Client " << name() << " " << own_addr
                                << " sending Peer SensorNet Cue of size
" << track_size << " bits\n";
                        cMessage *work = new cMessage( name());
                        work->setLength(track_size);
                        work->addPar("src") = own_addr;
                        work->addPar("track") = i;
                        work->setTimestamp(); //puts a current
time timestamp on it.
                        send( work, "Cueout" );
                }
        }

        for(;;)

```



```

    {
        cMessage *msg = receive();
        // Make sure you put in some delay for handling
of the message
        wait(process_time);
        avg_utilization = avg_utilization + process_time;
        resp_v.record(avg_utilization/simTime());

        //send the track list out that we just received,
as though it was ours.
        send(msg, "out");
    }
}

```

```

//-----
----
// file: SCA.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 3 Dec 2003
// This is a generic Sensor Controlling Authority.
//-----
----

```

```

#include "omnetpp.h"

```

```

class SCA : public cSimpleModule
{
    Module_Class_Members(SCA,cSimpleModule,16384)
    virtual void activity();
};

```

```

Define_Module( SCA );

```

```

void SCA::activity()

```

```

{
    double        process_time        =        parentModule()-
>par("Process_Time");

    for(;;)
    {
        // receive msg (implicit queueing!)
        cMessage *msg = receive();
        // Make sure you put in some delay for handling
of the message
        wait(process_time);

        //here we'll measure how long it took to get the
cueing message here.

        //*****
        *****

    }
}

//-----
----

// file: SFP.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 25 Nov 2003
// A generic Sensor Fusion Processor.
//-----
----

#include "omnetpp.h"

class SFP : public cSimpleModule
{
    Module_Class_Members(SFP,cSimpleModule,16384)

```

```

        virtual void activity();
};

Define_Module( SFP );

void SFP::activity()
{
    int own_addr = gate( "SNout" )->toGate()->index();
    int track_size = 0;
    int      num_radar_sensors      =      parentModule()-
>par("num_RadarSensors");
    int      radar_track_size      =      parentModule()-
>par("RadarTrackSize");
    int ir_track_size = parentModule()->par("IRTrackSize");
    int num_tracks = parentModule()->par("num_Tracks");
    double delay = parentModule()->par("TrackDelay");
    double time_marker = 0.00000;
    int num_fr = parentModule()->par("num_FusionRequests");
    int fusion_marker = 0;
    // double avg_utilization = 0.0;
    double      process_time      =      parentModule()-
>par("Process_Time");
    // cOutVector resp_v("SFP Utilization");
    long pointer;
    cArray *pntr;
    // cArray tracks;

    int track_list[100];

    // for (int t=0; t<100;t++)
    // {
    //     track_list[t] = -1;
    // }

    for(;;)

```

```

{

    //here's the message for each target that needs to be
    sent to the SensorNet

    ev << "SFP simTime() = " << simTime() << '\n';
    ev << "SFP time_marker = " << time_marker << '\n';

    if(simTime()>=time_marker)
    {
        time_marker = simTime()+delay;
        fusion_marker = 0;

    ev << "SFP simTime() = " << simTime() << '\n';
    ev << "SFP time_marker = " << time_marker << '\n';

        for(int i=0;i<num_tracks;i++)
        {

            ev << "In SFP Module inside Track
Sending Loop" << '\n';

            if (true)//((track_list[i] == own_addr)
|| (track_list[i] == -1))
            {
                ev << "SFP Checking to see if it
should send out any tracks" << endl;
                if(i<=(num_radar_sensors-1))
                {
                    track_size
radar_track_size;
                }
                else
                {
                    track_size = ir_track_size;
                }
            }
        }
    }
}

```

```

//                                ev << "num_radar_sensors-1 =
" << num_radar_sensors-1 << '\n';
//                                ev << "i = " << i << '\n';
//                                ev << "if i is smaller, then
tracksize should be equal to radar_track_size" << '\n';
//                                ev << "track_size = " <<
track_size << '\n';

                                ev << name() << " " << own_addr <<
" sending track of size " << track_size
                                << " bits\n";
                                cMessage *work = new cMessage(
name());

                                work->addPar("src") = own_addr;
                                work->addPar("track") = i;
//tracks & targets are counted 0 to n-1.
                                work->setLength(track_size);
                                work->setTimestamp(); //puts a
current time timestamp on it.
                                send(work, "SNout");
                                }
                                else if(fusion_marker<num_fr)
                                {
                                ev << name() << " " << own_addr <<
" sending ColFus Request of size "
                                << track_size << " bits\n";
                                cMessage *request = new cMessage(
name());

                                request->addPar("src") = own_addr;
                                request->addPar("track") = i;
//tracks & targets are counted 0 to n-1.
                                request->addPar("fwd") = true;
                                request->setLength(50); //EMBEDDED
PARAMETER ***ColFus Request Size***
                                request->setTimestamp();
                                send(request, "SNRequestout");
                                }

```

```

        fusion_marker++;
    }
}
else
{
    wait(.001);
    ev << "SFP on standby" << endl;
}

//here's the message receiving/handling area.
// receive msg (implicit queueing!)
simtime_t timeout = delay;
cMessage *msg = receive(timeout);
ev << "SFP waiting to receive message" << endl;
// Make sure you put in some delay for handling
of the message
wait(process_time);

if (msg != NULL)
{
    if (msg->arrivedOn("SNRequestin"))
    {
        delete msg;
    }
    else if (msg->arrivedOn("TrackListin"))
    {
        pointer = msg->par("mtlp");
        ev << " Pointer in SFP Process is " <<
pointer << endl;

        pntr = (cArray *) pointer;
        cArray tracks = *pntr;
        for (int i=0; i < tracks.items(); i++)
        {
            track_list[i] = (int) tracks[i];

```

```

        ev << "Track List Entry " << i <<
" was assigned " << track_list[i] << endl;
        //This gives us the track_list
array for use above.
    }
}
}

}
}

```

```

//-----
----
// file: SFPIC.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 1 Dec 2003
// A SFP Interface Capsule within the SensorNet.
//-----
----

```

```

#include "omnetpp.h"

```

```

class SFPIC : public cSimpleModule
{
    Module_Class_Members(SFPIC,cSimpleModule,16384)
    virtual void activity();
};

```

```

Define_Module( SFPIC );

```

```

void SFPIC::activity()
{
    double avg_utilization = 0.0;
    double      process_time      =      parentModule()-
>par("Process_Time");

```

```

cOutVector resp_v("SFPIC Utilization");
int num_sfps = parentModule()->par("num_SFps");
int      fused_track_size      =      parentModule()-
>par("FusedTrackSize");

    ev << "In SFPIC Module at point 1" << '\n';
    //*****

int local_target_list[100];
for (int t=0; t<100;t++)
{
    local_target_list[t] = -1;
}
int request_registry[100];
for (int r=0; r<100;r++)
{
    request_registry[r] = -1;
}

// pntr = &trackids;
// pointer = (long) pntr;

    for(;;)
    {

        //Up here we need to send tracks to the Weapons
        Platforms!!

        //*****
        *****

        // receive msg (implicit queueing!)

```



```

        cMessage *msg = receive();

        // Make sure you put in some delay for handling
of the message
        wait(process_time);
        avg_utilization = avg_utilization + process_time;
        resp_v.record(avg_utilization/simTime());

        if (msg->arrivedOn("TrackListin"))
        {
            if (msg->hasPar("mtlp"))
            {
                for(int s=0; s<num_sfps-1; s++)
                {
                    cMessage *copy = (cMessage *) msg-
>dup();

                    send(copy, "TrackListout", s);
                    ev << "SFPIC sending Master Track
List to SFP " << s << '\n';
                }
                delete msg;
            }
            if (msg->hasPar("tlp"))
            {
                //read the target list into the local
target list.
            }
        }
    }

    else if (msg->arrivedOn("Shortin"))
    {
        //there's a bit more delay in doing this
operation than normal
        wait(process_time);

```

```

        avg_utilization      =      avg_utilization      +
process_time;

        resp_v.record(avg_utilization/simTime());

        int target = msg->par("target"); //this is
the target the shorting order is for.
        if(local_target_list[target] != -1)
        {
                bool active = msg->par("active");
//this allows a weapon platform to activate/inactivate a
short order.

                if(active)
                {
                        ev << "SFPIC Marking Target " <<
target << " for Shorting!" << '\n';
                        request_registry[target] = true;
                }
                else //elsewise it must have already
fired on the target.
                {
                        ev << "SFPIC Deactivating Target "
<< target << '\n';
                }
        }
        else
        {
                ev << "SFPIC Cannot Short Target " <<
target << "!!!" << '\n';
        }
    }
    else if (msg->arrivedOn("SFPin"))
    {
            int source = msg->par("src");
            int track = msg->par("track");
            if (request_registry[track] &&
(local_target_list[track] == source))

```

```

        //request registry having value of true
for a track means it is active.
    {
        cMessage *copy = (cMessage *) msg-
>dup();
        send(copy, "WPout");
        send(msg, "TCCout");
    }
else //there's no active request for it at
this point
    {
        send(msg, "TCCout");
    }
}
else if (msg->arrivedOn("SFPRequestin"))
{
    //if I'm smart enough I might be able to
figure out how to model the whole forum thing****

    //*****
    *****

    ev << "SFPIC got a ColFus Request" << endl;
    //who did it come from?
    int id = msg->par("src");

    msg->setLength(fused_track_size); //set the
length of the fused track
    send(msg, "SFPRequestout", id); //send the
fused track back to the requesting SFP
}
}
}

//-----
----
// file: TCC.cpp

```

```

// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 1 Dec 2003
// The Track Correlation Capsule within the SensorNet
//-----
----

#include "omnetpp.h"

class TCC : public cSimpleModule
{
    Module_Class_Members(TCC,cSimpleModule,16384)
    virtual void activity();
};

Define_Module( TCC );

void TCC::activity()
{
    double avg_utilization = 0.0;
    double      process_time      =      parentModule()-
>par("Process_Time");
    double fusion_time = parentModule()->par("Fusion");
    double list_time = parentModule()->par("ListCheck");
    int      fused_track_size      =      parentModule()-
>par("FusedTrackSize");
    cOutVector resp_v("TCC utilization");
    double num_tracks = parentModule()->par("num_Tracks");
    double      num_targets      =      parentModule()-
>par("num_Targets");
    int      num_radarsensors      =      parentModule()-
>par("num_RadarSensors");
    int      num_irsensors      =      parentModule()-
>par("num_IRSensors");
    double num_sensors = num_radarsensors+num_irsensors;
    int num_sfps = parentModule()->par("num_SFps");

```

```

double cue_variable = 0.000;
double random_num;
int dropped_tracks = 0;
int dropped_cues = 0;
int forwarded_tracks = 0;
int forwarded_cues = 0;
int total_tracks = 0; //total tracks received
int total_cues = 0;

ev << "In TCC Module at point 1" << '\n';
//*****

int track_list[100];
for (int t=0; t<100;t++)
{
    track_list[t] = -1;
}
for (t=0; t<num_tracks; t++)
{
    //in an effort to simplify the simulation, here
we'll designate the winning SFPs up front
    track_list[t] = uniform(0,num_sfps);
    ev << "The SFP for Track " << t << " is SFP " <<
track_list[t] << '\n';
}

for(;;)
{
    cMessage *msg = receive();
    // Make sure you put in some delay for handling
of the message
    wait(process_time);
    avg_utilization = avg_utilization + process_time;
    resp_v.record(avg_utilization/simTime());
}

```

```

        cue_variable = (1/num_sensors); //watch out for
divide by 0 errors

        if(total_cues<num_tracks) //this lets the first
track through for each actual object out there.
        {
            cue_variable = 1.000;
        }

        if (msg->arrivedOn("Cuein"))
        {
            random_num = uniform(.01,1); //we're going
to drop all but fusion variable % of Cues.

            //let's see what the variables
are!*****
            ev << "TrackCorrelationCapsule, seeing if
the cue is redundant" <<'\n';
            ev << "Here's the Random Number --> " <<
random_num <<'\n';
            ev << "Here's the Cue Variable --> " <<
cue_variable << '\n';
            //If the Random Number is larger, it's
correlated (dropped)
            //Elsewise, it's considered a valid cue and
returned as such

            if (random_num > cue_variable) //need to
drop the message and wait for the next one.
            {
                ev << "##TCC Dropping the Cue!##" <<
'\n';

                delete msg;
            //
                dropped_cues++;
            }
            else //validate the cue

```

```

        {

            ev << "##TCC Validated the Cue!##" <<
'\n';

            //Note, the actual cue correlation
request is passed to an internal capsule, clearing the
            //TCC to handle other incoming cues.
This is modeled by subtracting fusion_time
            //from the fused message's timestamp.
This shows the time delay in the end.
            simtime_t temp = msg->timestamp();
            if (temp>list_time)
            {
                temp = temp - list_time;
            }
            msg->setTimestamp(temp);

            ev << "TCC.Cueout --> CC.in" << '\n';
            // return the track to the Cue Capsule;
            send(msg, "Cueout");
//            forwarded_cues++;
        }
    }
    else if (msg->arrivedOn("SFPICin"))
    {
//        total_tracks++;
        //Instead of the fusion variable, we pre-
designate winners above and drop all others.

        ev << "@@TCC About to Process Message!@"
<<'\n';

        int source = msg->par("src");
        ev << "Here's the SFP the Track came from --
> " << source <<'\n';

```

```

        int track = msg->par("track");
        ev << "Here's the Track it pertains to --> "
<< track <<'\n';

        if (track_list[track] == source)
        {
            ev << "@@TCC Fusing the message!@" <<
'\n';

            //Note, the actual fusion request is
passed to an internal capsule, clearing the
            //TCC to handle other incoming tracks.
This is modeled by subtracting fusion_time
            //from the fused message's timestamp.
This shows the time delay in the end.
            simtime_t temp = msg->timestamp();
            if (temp>fusion_time)
            {
                temp = temp - fusion_time;
            }
            msg->setTimestamp(temp);

            // change the size to a fused track
size
            msg->setLength(fused_track_size);

            ev << "TCC --> TRC" << '\n';
            // forward the track for writing to the
Track Registry's master track list

            send(msg, "TRCout");
//            forwarded_tracks++;
        }
        else //drop the message
        {

```



```

                                ev << "@@TCC Dropping the message!@"
<< '\n';

                                delete msg;
//                                dropped_tracks++;
                                }
                                }
                                else if (msg->arrivedOn("TRCin"))
                                {
                                        ev << "TrackCorrelationCapsule processing
msg from TRCin" << '\n';
                                        //note, this causes minimal delay, as the
List Maintenance Capsule is essentially a
                                        //Sensor Fusion Processor, complete with a
Track List Capsule that has an active and
                                        //semi-active Track Registry. The new list
goes to the inactive Track Registry
                                        //then, after the inactive comes on line, it
is given to the formerly active TR.

                                        //This is an estimated service delay to
switch between the active and semi-active TRs.
                                        wait(process_time);
                                        avg_utilization      =      avg_utilization      +
process_time;
                                        resp_v.record(avg_utilization/simTime());

                                        delete msg;
                                }
                                else if (msg->arrivedOn("PHICPSNin"))
                                {
                                        /**Here we process the Abstracted master
track list and send out cues.
                                        }
                                else if (msg->arrivedOn("PHICBMC2in"))
                                {
                                        //We route the target assignment messages
directly to the TRC

```

```

        send(msg, "TRCout");
    }
    else
    {
        ev << "***ERROR: TCC did not handle
message!***";
    }
}
}

```

```

//-----
----

```

```

// file: TRC.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 29 Nov 2003
// The Track Registry Capsule keeps track of the master
track list
// for the SensorNet.
//-----
----

```

```

#include "omnetpp.h"

```

```

class TRC : public cSimpleModule
{
    Module_Class_Members(TRC,cSimpleModule,16384)
    virtual void activity();
};

```

```

Define_Module( TRC );

```

```

void TRC::activity()
{
    double delay = parentModule()->par("TrackListDelay");

```

```

double time_marker = 0.00000;
int num_tracks = parentModule()->par("num_Tracks");
int      fused_track_size      =      parentModule()-
>par("FusedTrackSize");
double avg_utilization = 0.0;
double      process_time      =      parentModule()-
>par("Process_Time");
cOutVector resp_v("TRC Utilization");

```

```

ev << "In TRC Module at point 1" << '\n';
//*****

```

```

int track_list[20][100];
for (int a=0; a<20;a++)
{
    for (int b=0; b<100;b++)
    {
        track_list[a][b] = -1;
    }
}

```

```

int target_list[20][100];
for (int c=0; c<20;c++)
{
    for (int d=0; d<100;d++)
    {
        target_list[c][d] = -1;
    }
}

```

```

int x = 0; //This is the counter that keeps the track
and target lists synchronized.

```

```

int *pntrA;
int *pntrB;

```

```

long master_track_list_ptr;
long target_list_ptr;

    for(;;)
    {

//here's the Master Track List and Target List messages
going out
        if(simTime()>time_marker)
        {
            cMessage *trackmsg = new cMessage( name());
            cMessage *targetmsg = new cMessage( name());

//            pnterA = &track_list[a][0];
//            master_track_list_ptr = (long) pnterA;
//            trackmsg->addPar( "mtlp" );
//            trackmsg->par("mtlp") =
master_track_list_ptr;

//make the pointer to the current Master
Track List
            char * master_track_list_ptr;
            master_track_list_ptr = (char *)
track_list[a];
            trackmsg->addPar( "mtlp" );
            trackmsg->par("mtlp") =
master_track_list_ptr;

//make the pointer to the current Target
List
            char * target_list_ptr;
            target_list_ptr = (char *) target_list[a];
            targetmsg->addPar( "tlp" );
            targetmsg->par("tlp") = target_list_ptr;

//send out the Master Track List

```

```

        cMessage *copy = (cMessage *) trackmsg-
>dup();

        send(copy, "TSCout");
        ev << "Sending Master Track List to TSC with
Track List " << x << '\n';
        send(trackmsg, "TCCout");
        ev << "Sending Master Track List to TCC with
Track List " << x << '\n';

        //send out the Target List
        send(targetmsg, "TSCout");
        ev << "Sending Target List to TSC" << '\n';

        time_marker = simTime()+delay;

        if(x>19)
        {
            ev << "TRC switching to Track and
Target Lists " << x << endl;
            x = 0;
            //Here's where we copy, then move to
the next Master Track List
            for (int m=0; m<100; m++)
            {
                track_list[x][m] =
track_list[20][m];
            }
            //Here's where we copy, then move to
the next Target List
            for (int n=0; n<100; n++)
            {
                track_list[x][n] =
track_list[20][n];
            }
        }
        else

```

```

        {
            x++;
            ev << "TRC switching to Track and
Target Lists " << x << endl;
            //Here's where we copy, then move to
the next Master Track List
            for (int m=0; m<100; m++)
            {
                track_list[x][m] = track_list[x-
1][m];
            }
            //Here's where we copy, then move to
the next Target List
            for (int n=0; n<100; n++)
            {
                track_list[x][n] = track_list[x-
1][n];
            }
        }
    }

//here's the message receiving/handling area.
    // receive msg (implicit queueing!)
    simtime_t timeout = .01;
    cMessage *msg = receive(timeout);
    // Make sure you put in some delay for handling
of the message
    wait(process_time);
    avg_utilization = avg_utilization + process_time;
    resp_v.record(avg_utilization/simTime());

    if (msg != NULL)
    {
        if (msg->arrivedOn("TCCin"))
        {

```

```

        //here's where we need to do the magic
of making the Master Track List and Target List

        if(msg->hasPar("BMC2_wp"))           //this
tells us if it's a target assignment message
        {
            ev << "Target Message Received!"

<< '\n';

            int tg = msg->par("target");
            int wp = msg->par("BMC2_wp");
            target_list[x][tg] = wp;
        }
        if(msg->hasPar("track"))
        {
            ev << "Track Message Received!" <<
'\n';

            int source = msg->par("src");
            int trk = msg->par("track");
            track_list[x][trk] = source;
        }
        ev << "Track/Target Message Deleted!"

<< '\n';

        delete msg;
    }
    else
    {
        ev << "ERROR in the TRC Capsule!!!" <<
'\n';
    }
}

}

}

//-----
----
// file: TSC.cpp
// author: Joel D. Babbitt

```

```

// Thesis Work @ NPS
// Date: 29 Nov 2003
// The Track Server Capsule serves the master track list to
all
//-----
----

#include "omnetpp.h"

class TSC : public cSimpleModule
{
    Module_Class_Members(TSC,cSimpleModule,16384)
    virtual void activity();
};

Define_Module( TSC );

void TSC::activity()
{
    double avg_utilization = 0.0;
    double      process_time      =      parentModule()-
>par("Process_Time");
    cOutVector resp_v("TSC Utilization");
    int num_tracks = parentModule()->par("num_Tracks");
    int      fused_track_size      =      parentModule()-
>par("FusedTrackSize");

    ev << "In TSC Module at point 1" << '\n';
    //*****

    for(;;)
    {
        cMessage *msg = receive();
        // Make sure you put in some delay for handling
of the message

```



```

wait(process_time);
avg_utilization = avg_utilization + process_time;
resp_v.record(avg_utilization/simTime());

if (msg->arrivedOn("WPICin"))
{
    //For purposes of this simulation, the
actual shorting is being handled in the SFPIC
    //The reason for this is because the data
structures needed to model it here are beyond
    //my meager programming skills
    send(msg, "Shortout");
}
else if (msg->arrivedOn("TRCin"))
{
    if (msg->hasPar("mtlp"))
    {
        //Here we distribute the Master Track
List to the SFPIC, the BMC2,
        //and an abstracted version to all peer
sensor nets (simulated as
        //a smaller message size).
        cMessage *copy = (cMessage *) msg-
>dup();
        cMessage *abstract = (cMessage *) msg-
>dup();
        copy-
>setLength(num_tracks*fused_track_size);
        send(copy, "TrackListout");
        ev << "TSC Forwarding the Master Track
List to the SFPIC" << '\n';
        abstract-
>setLength(num_tracks*fused_track_size*.5);
        send(abstract, "PSNout");
        ev << "TSC Forwarding the Master Track
List to the Peer Sensor Nets" << '\n';

```

```

        msg-
>setLength(num_tracks*fused_track_size);
        send(msg, "BMC2out");
        ev << "TSC Forwarding the Master Track
List to the BMC2" << '\n';
    }
    else if (msg->hasPar("tlp"))
    {
        //Here we distribute the Target List to
the BMC2 and SFPIC.
        //As the MTL and TL are really the same
list, this has no size.
        //The size of this message is already
encapsulated into the
        //Master Track List.
        cMessage *copy = (cMessage *) msg-
>dup();
        send(copy, "TrackListout");
        ev << "TSC Forwarding the Target List
to the SFPIC for Shorting Orders" << '\n';
        send(msg, "BMC2out");
        ev << "TSC Forwarding the Target List
to the BMC2" << '\n';
    }
    else
    {
        ev << "ERROR in the TSC!!" << endl;
    }
}
}
}

```

```

//-----
----

```

```

// file: WP.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS

```

```

// Date: 3 Dec 2003
// A Weapon Platform that connects to the SensorNet
//-----
----

#include "omnetpp.h"

class WP : public cSimpleModule
{
    Module_Class_Members(WP,cSimpleModule,16384)
    virtual void activity();
};

Define_Module( WP );

void WP::activity()
{

    ev << "In WP Module at point 1" << '\n';
    //*****

    double          process_time          =          parentModule()-
>par("Process_Time");
    //  int  BMC2_own_addr  =  gate(  "BMC2in"  )->toGate()-
>index();
    //  int WPIC_own_addr = gate( "out" )->toGate()->index();

    ev << "In WP Module at point 2" << '\n';
    //*****

    for(;;)
    {

        // receive msg (implicit queueing!)
        cMessage *msg = receive();

```

```

        // Make sure you put in some delay for handling
of the message
        wait(process_time);

        if (msg->arrivedOn("BMC2in"))
        {
/*          int wp = msg->par("BMC2_wp");
          if (BMC2_own_addr == wp)
          {
              msg->addPar("WPIC_wp") = WPIC_own_addr;
              send(msg, "out"); //forward it to the
WPIC.
          }
          else
          {
              ev << "ERROR: Weapon Platform " <<
BMC2_own_addr << " received WP "
              << wp << "'s Firing Order!" <<
'\n';
          }
*/
        }
        if (msg->arrivedOn("in"))
        {
            //we need to measure here how long it took
to get a firing solution to the weapon platform
            //this time measurement would include from
SFPIC to WP.
        }
    }
}

//-----
----

// file: WPIC.cpp
// author: Joel D. Babbitt
// Thesis Work @ NPS
// Date: 2 Dec 2003

```

```

// The Weapon Platform Interface Capsule within the
SensorNet.

//-----

----

#include "omnetpp.h"

class WPIC : public cSimpleModule
{
    Module_Class_Members(WPIC,cSimpleModule,16384)
    virtual void activity();
};

Define_Module( WPIC );

void WPIC::activity()
{
    double avg_utilization = 0.0;
    double      process_time      =      parentModule()-
>par("Process_Time");
    cOutVector resp_v("WPIC utilization");

    ev << "In WPIC Module at point 1" << '\n';
    /*******

    for(;;)
    {
        // receive msg (implicit queueing!)
        cMessage *msg = receive();
        // Make sure you put in some delay for handling
of the message
        wait(process_time);
        avg_utilization = avg_utilization + process_time;
        resp_v.record(avg_utilization/simTime());

```

```

    if (msg->arrivedOn("WPin"))
    {
        send(msg, "TrackRequestout");
    }
    else if (msg->arrivedOn("TrackRequestin"))
    {
        int dest = msg->par("WPIC_wp");
        send(msg, "WPout", dest);
    }
}
}

```

APPENDIX G. SIMULATION DATA

	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)						
5 Tracks			SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
56.6 Kbps	0.13	0.06	0.0002	0.0005	0.027	0.00008	0.0001	0.00009
1.44 Mbps	0.13	0.055	0.0002	0.0004	0.027	0.00008	0.0001	0.00009
120 Mbps	0.13	0.055	0.00022	0.00045	0.027	0.00008	0.0001	0.00009
50 Tracks								
56.6 Kbps	0.27	0.21	0.0022	0.0045	0.27	0.0008	0.0006	0.0005
1.44 Mbps	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
120 Mbps	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
100 Tracks								
56.6 Kbps	0.46	0.4	0.0046	0.009	0.53	0.0016	0.0011	0.0009
1.44 Mbps	0.46	0.4	0.0046	0.009	0.53	0.0016	0.0011	0.0009
120 Mbps	0.46	0.4	0.0046	0.009	0.53	0.0016	0.0011	0.0009
500 Tracks								
56.6 Kbps	16	11	0.022	0.032	0.92	0.002	0.001	0.0008
1.44 Mbps	16	11	0.022	0.032	0.92	0.002	0.001	0.0008
120 Mbps	16	11	0.022	0.032	0.92	0.002	0.001	0.0008

Constants

Track Size: 512/1024/1M bits Radar Delay: .5 sec
 Radars: 4 IR Delay: 2 sec IR Sensors: 2
 Tracked Objects: 50 Capsule Data Rate: 1 Gbps Collaboration Requests: 1
 Master Track List BC: 0.1sec Module Track Handling Time: 0.000005 sec
 Fusing Time: 0.01 sec

Table 1. Varying Data Rates

	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)						
5 Tracks			SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
1024 bits	0.13	0.055	0.0002	0.0004	0.027	0.00008	0.0001	0.00009
512 bits	0.13	0.055	0.0002	0.0004	0.027	0.00008	0.0001	0.00009
50 Tracks								
1024 bits	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
512 bits	0.27	0.21	0.0023	0.005	0.27	0.00082	0.00058	0.0005
100 Tracks								
1024 bits	0.46	0.4	0.0046	0.009	0.53	0.0016	0.0011	0.0009
512 bits	0.46	0.4	0.005	0.009	0.53	0.0016	0.0011	0.00095
500 Tracks								
1024 bits	16	11	0.022	0.032	0.92	0.002	0.001	0.0008
512 bits	16	11	0.023	0.031	0.94	0.0023	0.001	0.00085

Constants

Data Rates: 56.6Kbps/1.44Mbps/120 Mbps Radar Delay: .5 sec
 Radars: 4 Radar Delay: .5 sec IR Delay: 2 sec
 IR Sensors: 2 Capsule Data Rate: 1 Gbps Collaboration Requests: 1
 Master Track List BC: 0.1sec Fusing Time: 0.01 sec
 Module Track Handling Time: 0.000005 sec Track List Check Time: 0.0005 sec

Table 2. Varying Track Message Sizes

Varying Data Rates and Track Message Size

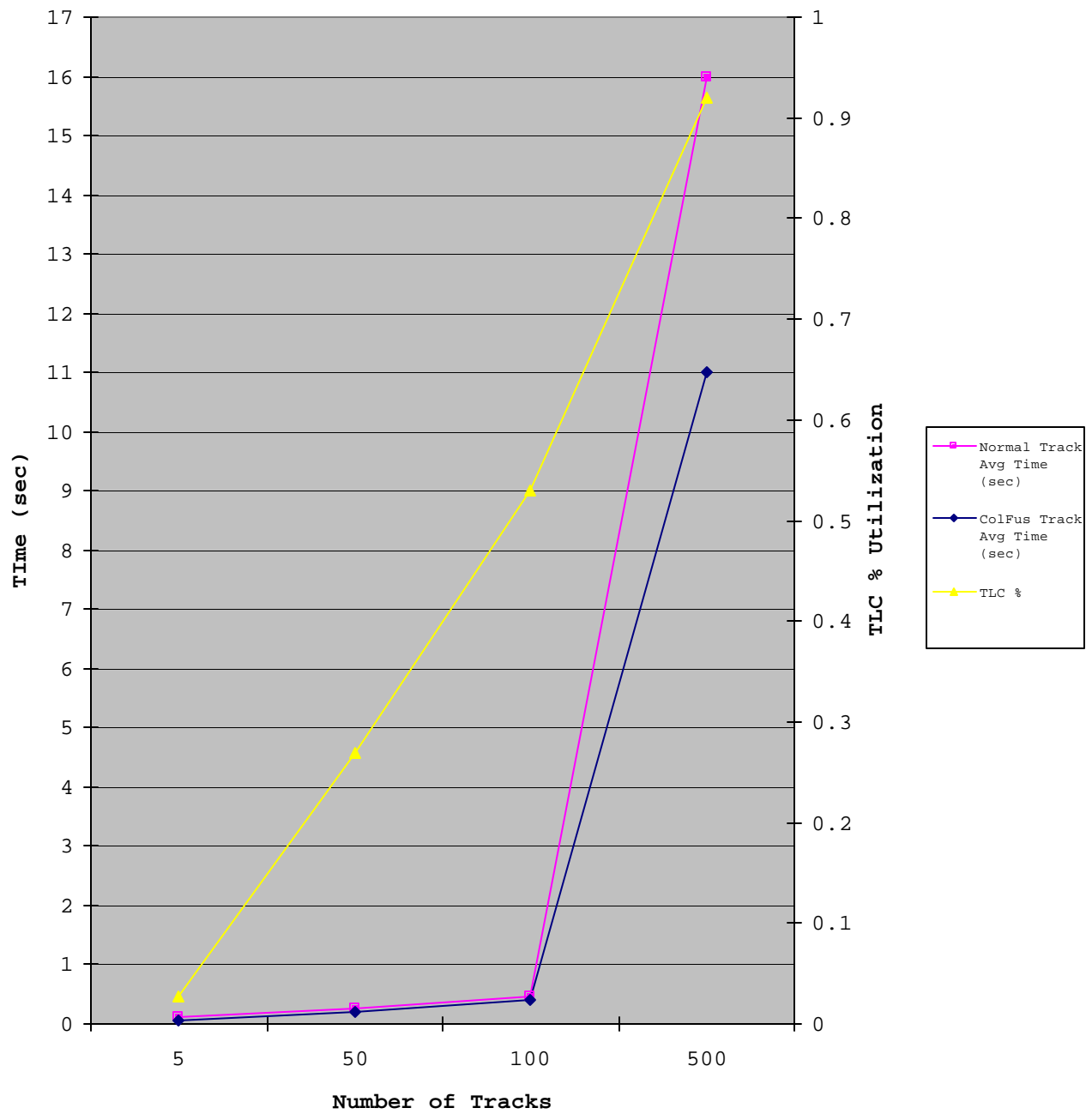


Figure 44. Varying Data Rates and Track Message Sizes

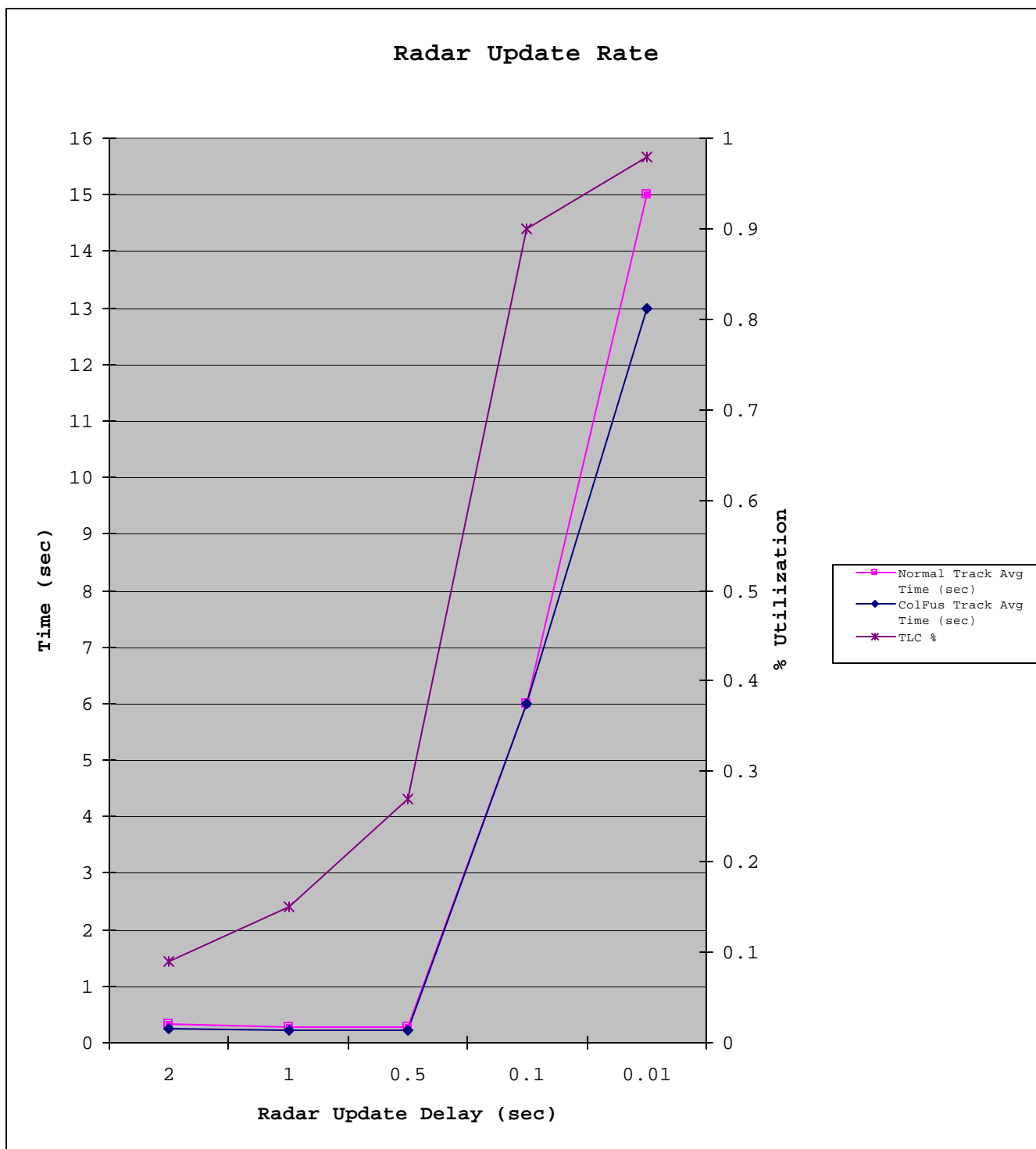


Figure 45. Ground-based Radar Update Delay

Table 3. Ground-based Radar Update Delay

Radar Delay (sec)	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
2	0.34	0.24	0.0007	0.0015	0.09	0.0003	0.00025	0.0002
1	0.29	0.22	0.0013	0.0025	0.15	0.00045	0.00035	0.0003
0.5	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
0.1	6	6	0.01	0.018	0.9	0.0025	0.0015	0.0013
0.01	15	13	0.1	0.011	0.98	0.0018	0.00025	0.0003

Constants

Data Rates: 1.44 Mbps Track Size: 1024 bits Radars: 4
 IR Delay: 2 sec IR Sensors: 2 Tracked Objects: 50
 Capsule Data Rate: 1 Gbps Collaboration Requests: 1
 Master Track List BC: 0.1sec Module Track Handling Time: 0.000005 sec
 Track List Check Time: 0.0005 sec Fusing Time: 0.01 sec

Table 4. Space-Based IR Update Delay

IR Delay	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
5	0.225	0.205	0.0021	0.0042	0.25	0.0009	0.0006	0.0005
2	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
1	0.34	0.23	0.0025	0.005	0.3	0.0009	0.00062	0.00055
0.5	0.42	0.25	0.003	0.006	0.35	0.001	0.0007	0.00061
0.1	0.55	0.26	0.007	0.014	0.81	0.0022	0.0014	0.0013
0.01	15	2.5	0.05	0.06	0.95	0.0017	0.00035	0.00033

Constants

Data Rates: 1.44 Mbps Track Size: 1024 bits Radar Delay: .5 sec
 Radars: 4 IR Sensors: 2 Tracked Objects: 50
 Capsule Data Rate: 1 Gbps Collaboration Requests: 1
 Master Track List BC: 0.1sec Module Track Handling Time: 0.000005 sec
 Track List Check Time: 0.0005 sec Fusing Time: 0.01 sec

Space-Based IR Update Rate

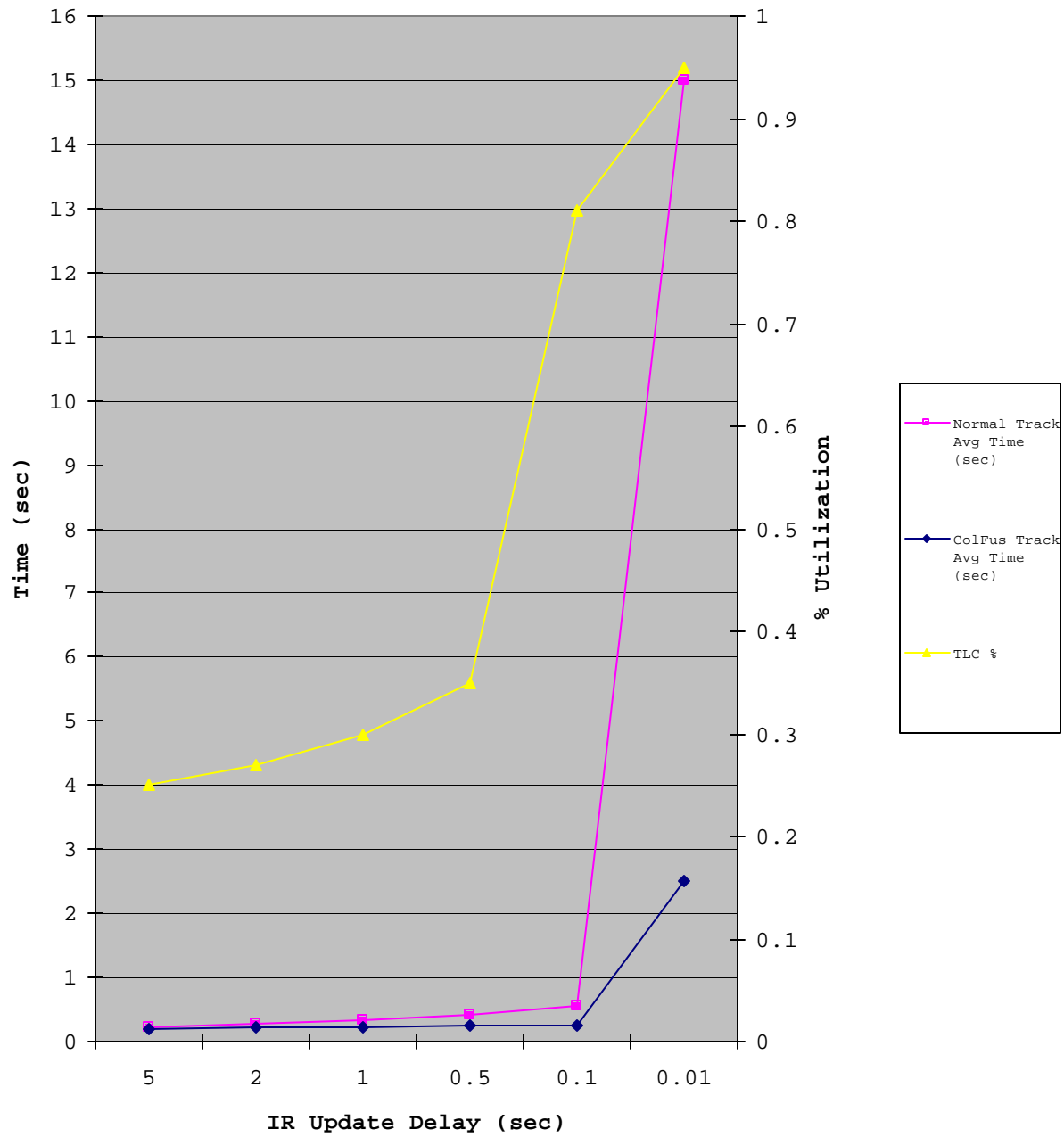


Figure 46. Space-Based IR Update Delay

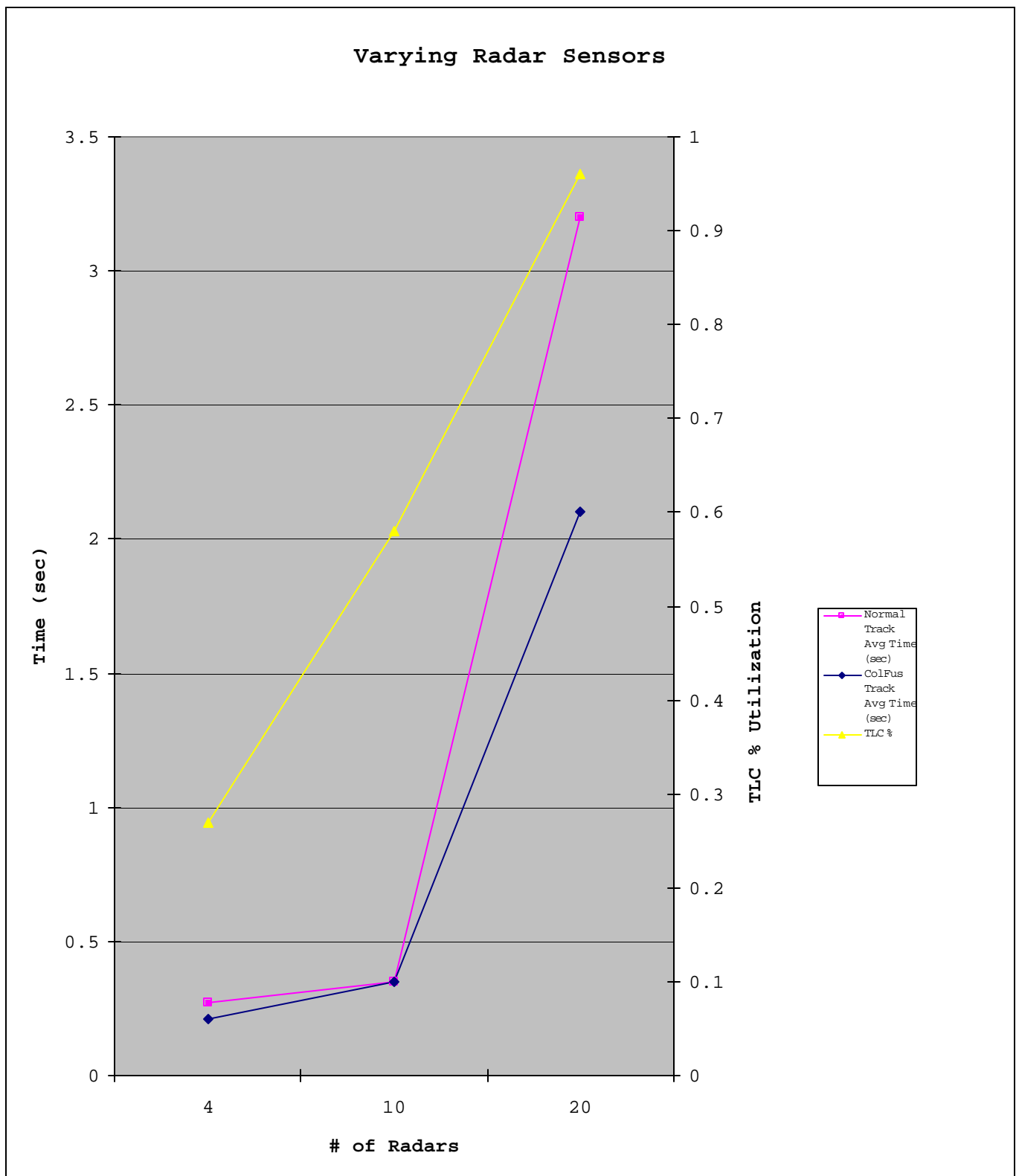


Figure 47. Varying Number of Ground-based Radar Sensors

Table 5. Varying Number of Ground-based Radar Sensors

Radar #	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
4	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
10	0.35	0.35	0.0055	0.01	0.58	0.0008	0.00053	0.00047
20	3.2	2.1	0.01	0.019	0.96	0.0007	0.0004	0.00038

Constants

Data Rates: 1.44 Mbps Track Size: 1024 bits Radar Delay: .5 sec
 IR Delay: 2 sec IR Sensors: 2 Tracked Objects: 50
 Capsule Data Rate: 1 Gbps Collaboration Requests: 1
 Master Track List BC: 0.1sec Module Track Handling Time: 0.000005 sec
 Track List Check Time: 0.0005 sec Fusing Time: 0.01 sec

Table 6. Varying Number of Space-based IR Sensors

IR #	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
1	0.25	0.21	0.0022	0.00453	0.25	0.001	0.00068	0.00057
2	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
4	0.34	0.22	0.0025	0.005	0.28	0.00065	0.0005	0.00042
10	0.52	0.26	0.0034	0.0065	0.35	0.00045	0.00032	0.00029

Constants

Data Rates: 1.44 Mbps Track Size: 1024 bits Radars: 4
 Radar Delay: .5 sec IR Delay: 2 sec Tracked Objects: 50
 Capsule Data Rate: 1 Gbps Collaboration Requests: 1
 Master Track List BC: 0.1sec Module Track Handling Time: 0.000005 sec
 Track List Check Time: 0.0005 sec Fusing Time: 0.01 sec

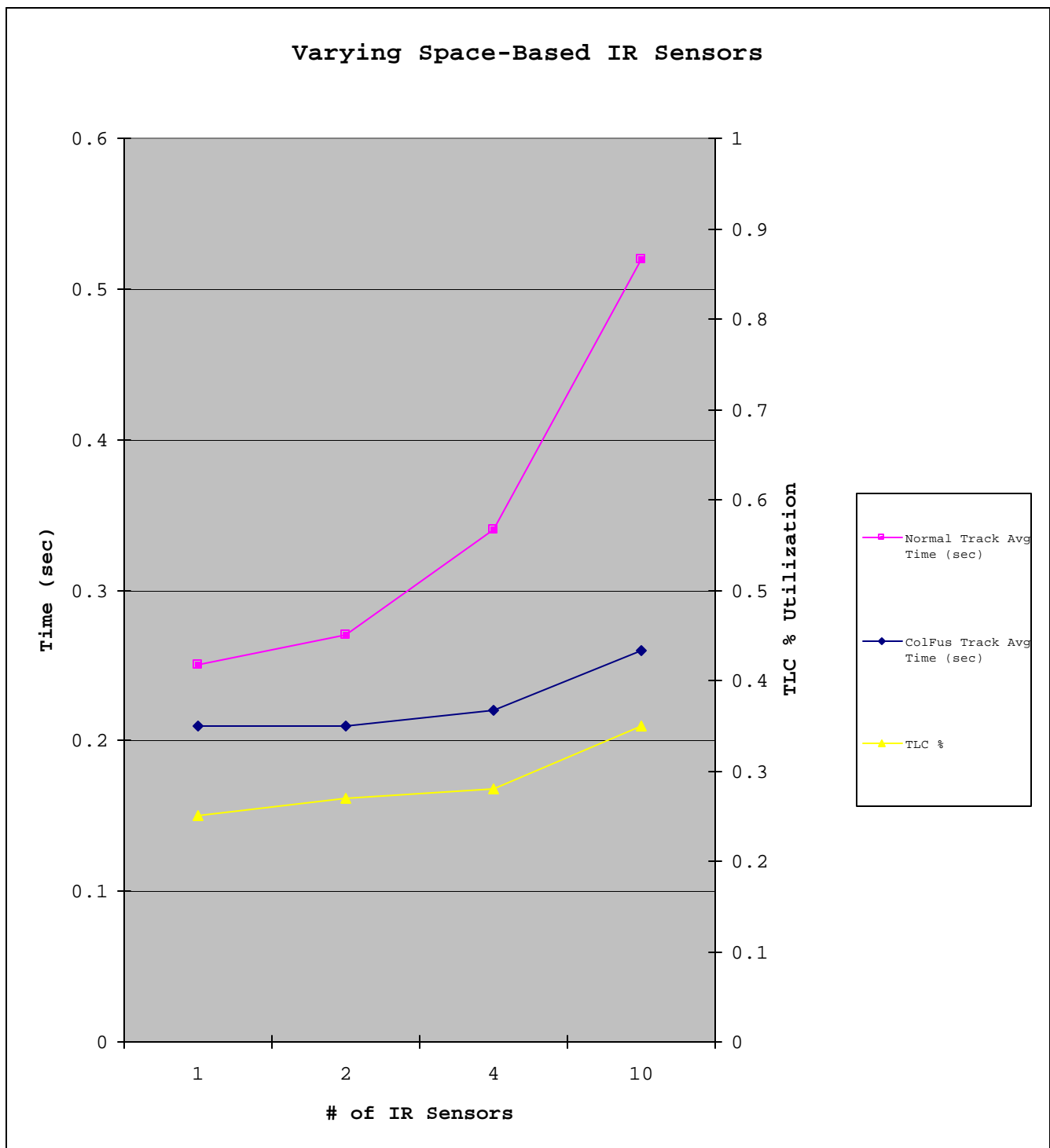


Figure 48. Varying Number of Space-based IR Sensors

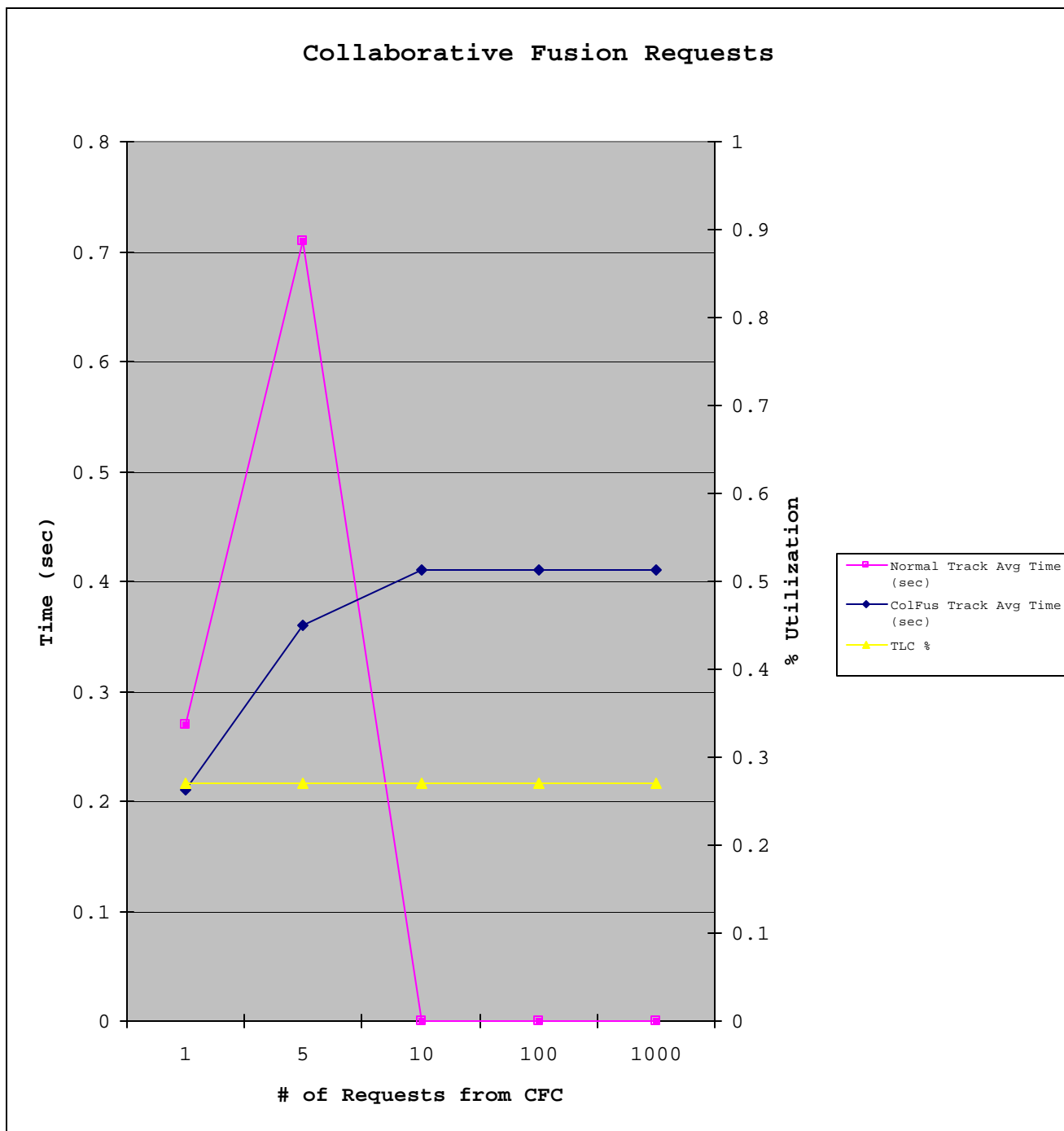


Figure 49. Collaborative Fusion Requests

Table 7. Collaborative Fusion Requests

# of Collaborative Fusion Requests from CFC	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
1	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
5	0.71	0.36	0.0021	0.005	0.27	0.0011	0.0012	0.0008
10	0	0.41	0.0021	0.0045	0.27	0.0011	0.0012	0.0008
100	0	0.41	0.0021	0.0045	0.27	0.0011	0.0012	0.0008
1000	0	0.41	0.0021	0.0045	0.27	0.0011	0.0012	0.0008

Constants

Data Rates: 1.44 Mbps Track Size: 1024 bits Radars: 4
 Radar Delay: .5 sec IR Sensors: 2 IR Delay: 2 sec
 Tracked Objects: 50 Capsule Data Rate: 1 Gbps
 Module Track Handling Time: 0.000005 sec Master Track List BC: 0.1sec
 Track List Check Time: 0.0005 sec Fusing Time: 0.01 sec

Table 8. Module Processing Time

Time each Module Handles a Track	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
0.000005	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
0.0005	0.34	0.21	0.0023	0.0045	0.53	0.08	0.059	0.051
0.05	22	6	0.99	0.99	0.99	0.3	0.6	0.5
	(17sec)	(15sec)						

Constants

Data Rates: 1.44 Mbps Track Size: 1024 bits Radar Delay: .5 sec
 Radars: 4 IR Delay: 2 sec IR Sensors: 2
 Tracked Objects: 50 Capsule Data Rate: 1 Gbps Collaboration Requests: 1
 Master Track List BC: 0.1sec Track List Check Time: 0.0005 sec
 Fusing Time: 0.01 sec

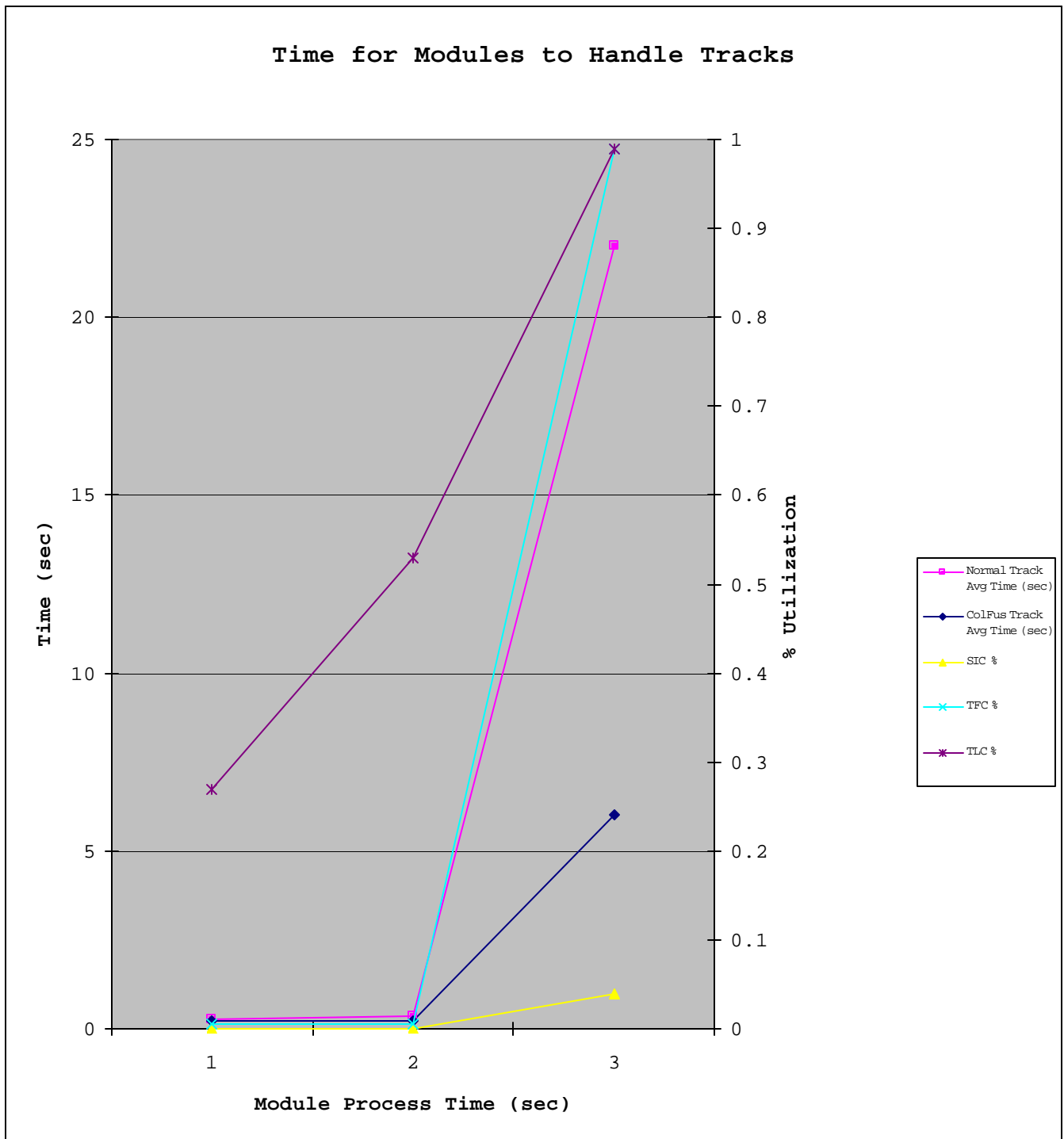


Figure 50. Module Processing Time

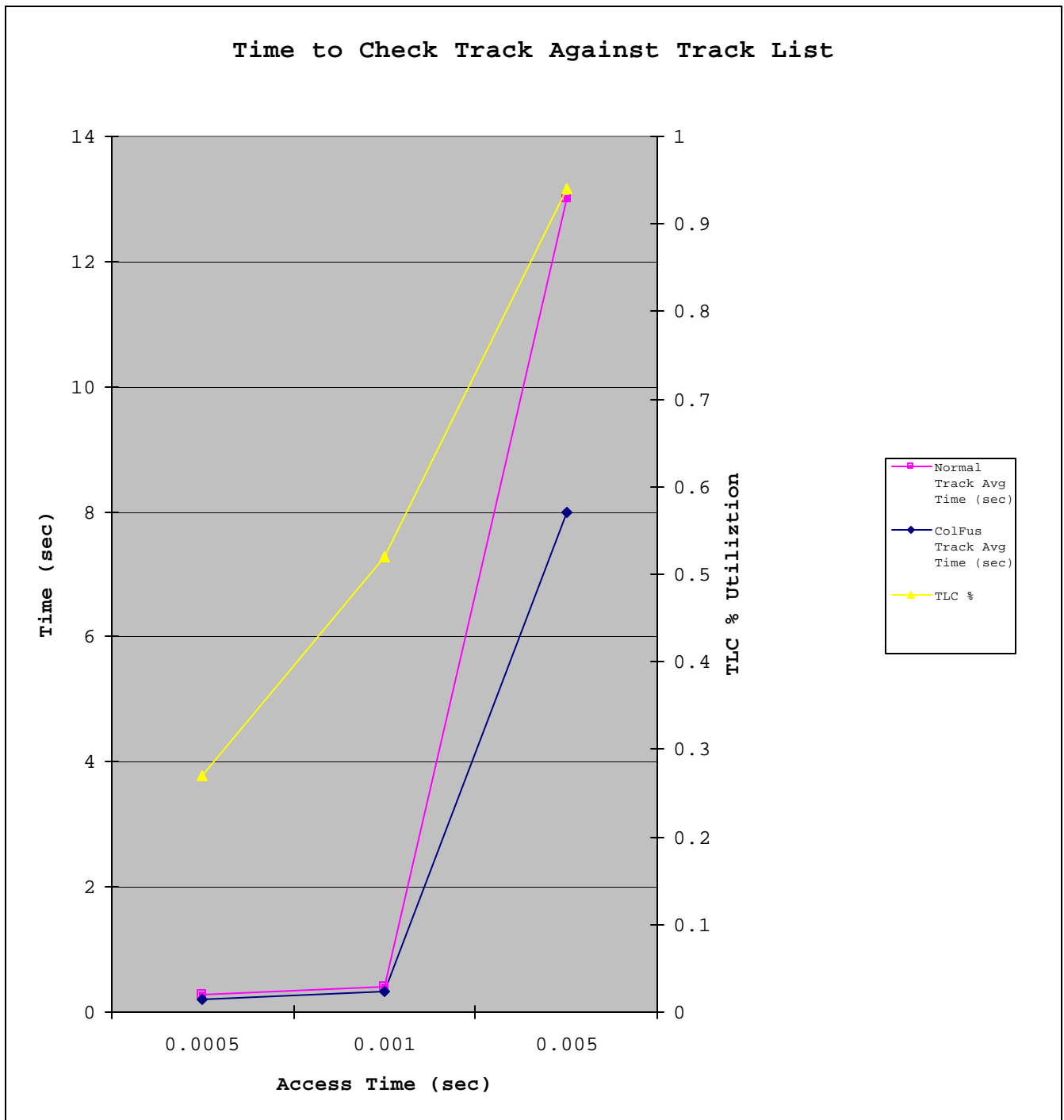


Figure 51. Track List Access Time

Table 9. Track List Access Time

Time to Check Track Against List	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
0.0005	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
0.001	0.4	0.33	0.0023	0.0045	0.52	0.00082	0.0006	0.0005
0.005	13	8	0.0023	0.003	0.94	0.00022	0.00014	0.00012

Constants

Data Rates: 1.44 Mbps Track Size: 1024 bits Radar Delay: .5 sec
 Radars: 4 IR Delay: 2 sec IR Sensors: 2
 Tracked Objects: 50 Capsule Data Rate: 1 Gbps Collaboration Requests: 1
 Master Track List BC: 0.1sec Module Track Handling Time: 0.000005 sec
 Fusing Time: 0.01 sec

Table 10. Time to Perform Track Fusion

Time to Perform Fusion	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
0.01	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
0.05	0.75	0.85	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
0.1	6.5	8	0.0023	0.0045	0.27	0.0005	0.00035	0.0003

Constants

Data Rates: 1.44 Mbps Track Size: 1024 bits Radar Delay: .5 sec
 Radars: 4 IR Delay: 2 sec IR Sensors: 2
 Tracked Objects: 50 Capsule Data Rate: 1 Gbps Collaboration Requests: 1
 Master Track List BC: 0.1sec Module Track Handling Time: 0.000005 sec
 Track List Check Time: 0.0005 sec

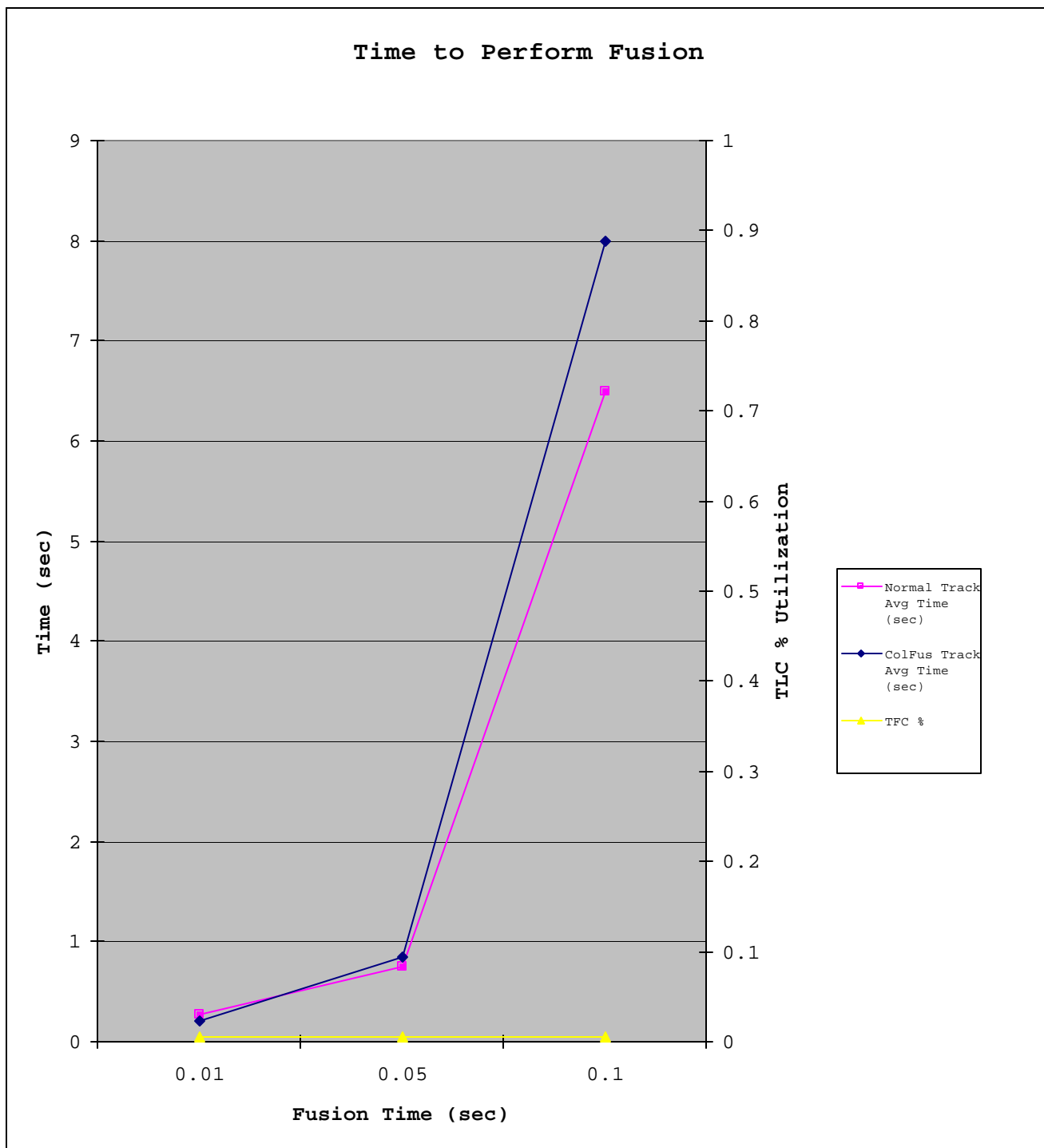


Figure 52. Time to Perform Track Fusion

Table 11. Master Track List Broadcast Times

Delay Between Master Track List Broadcasts	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
10	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.00055	0.00045
1	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.00055	0.00045
0.1	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
0.01	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0011	0.00097
0.001	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0056	0.0054

Constants

Data Rates: 1.44 Mbps Track Size: 1024 bits Radars: 4
 Radar Delay: .5 sec IR Sensors: 2 IR Delay: 2 sec
 Tracked Objects: 50 Capsule Data Rate: 1 Gbps Collaboration Requests: 1
 Module Track Handling Time: 0.000005 sec
 Track List Check Time: 0.0005 sec Fusing Time: 0.01 sec

Table 12. Capsule Data Rate

Data Rate Between Capsules	Normal Track Avg Time (sec)	ColFus Track Avg Time (sec)	SIC %	TFC %	TLC %	CFC %	SNIC %	SN.SFP %
1E+11	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
1000000000	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
1000000	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005
1000	0.27	0.21	0.0023	0.0045	0.27	0.00082	0.0006	0.0005

Constants

Data Rates: 1.44 Mbps Track Size: 1024 bits Radars: 4
 Radar Delay: .5 sec IR Delay: 2 sec IR Sensors: 2
 Tracked Objects: 50 Collaboration Requests: 1
 Master Track List BC: 0.1sec Module Track Handling Time: 0.000005 sec
 Track List Check Time: 0.0005 sec Fusing Time: 0.01 sec

LIST OF REFERENCES

A New Paradigm for Requirements Specification and Analysis of System-of-Systems, Caffall, Dale Scott. and James Bret Michael. Wirsing, M., Balsamo, S., and Knapp, A., eds., Lecture Notes in Computer Science: Proc. Monterey Workshop 2002: Radical Innovations of Software and Systems Engineering in the Future, Berlin: Springer-Verlag, 2003.

Battle Management, Haim Baruch, AIAA, 2000.

Charting the Seas of Information Technology, The Standish Group, 1994.

Conceptual Framework Approach for System-of-Systems Software Developments, NPS Thesis, Dale Scott Caffall, Mar. 2003.

Concurrency Control and Recovery in Database Systems Philip Bernstien, Vassos Hadzilacos, and Nathan Goodman, Addison-Wesley, New York, 1987.

Designing Concurrent, Distributed, and Real-Time Applications with UML, Hassan Gomaa, Addison-Wesley, New York, 2001.

Distributed Systems Concepts and Design, George Coulouris, Jean Dollimore, and Tim Kindberg, Addison-Wesley, New York, 2001.

Evolving A Simulation Model Product Line Software Architecture From Heterogeneous Model Representations, NPS Ph.D. Dissertation, Kevin J. Greaney, Sept. 2003.

Harnessing the Power of Technology, The Road to Ballistic Missile Defense From 1987-2007, BMDO, Sept. 2000.

HIPO: A Design Aid and Documentation Technique, IBM Corporation, Armonk, NY, 1974.

<http://www.fas.org/spp/starwars/program/dote99/99sbirs.htm>.

<http://www.omnetpp.org>.

<http://www.whitehouse.gov>, Press release 13 Dec. 2001.

<http://www.whitehouse.gov>, Press release 17 Dec. 2002.

Managing Software Requirements, Dean Leffingwell, Don Widrig, Addison-Wesley, 2000.

MDA Exhibit R-2 RDT&E Budget Item Justification (PE 0603889C), DTIC, Feb. 2003.

Naval Forces Capability for Theater Missile Defense, Naval Studies Board National Research Council, National Academies Press, 2001.

Office of the Secretary of Defense, SecDef Memo dated 2 Jan. 2002.

Pattern-Oriented Software Architecture: A System of Patterns, F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, John Wiley & Sons, New York, 1996.

Schaum's Outline UML, Simon Bennett, John Skelton, and Ken Lunn, McGraw-Hill, London, 2001.

Test and Evaluation of the Ballistic Missile Defense System FY 03 Progress Report, James Bret Michael, Phillip Pace, Man-Tak Shing, Murali Tummala and others, eds., Naval Postgraduate School, Sep 2003.

The UML Reference Manual, Grady Booch, Jim Rumbaugh, and Ivar Jacobson, Addison Wesley 1999.

Using UML for Modeling Complex Real Time Systems, Bran Selic and Jim Rumbaugh, April 1998.

BIBLIOGRAPHY

Air, Land, and Sea Application (ALSA) Center. *TADIL J, Introduction to Tactical Digital Information Link J and Quick Reference Guide*. June 2000.

<http://www.adtdl.army.mil>, Accessed November 2003.

Ballistic Missile Defense Organization. *Harnessing the Power of Technology: The Road to Ballistic Missile Defense from 1983-2007*. September 2000.

<http://www.acq.osd.mil/bmdo/bmdolink/pdf/power.pdf>,

Accessed November 2003.

Barach, Haim. 2001: Battle Management. *Theater Ballistic Missile Defense, Volume 192, Progress in Astronautics and Aeronautics*. American Institute of Aeronautics and Astronautics, Inc. 205-217.

Bennett, Simon, John Skelton, and Ken Lunn. *Schaum's Outline of UML*. London: McGraw-Hill, 2001.

Bernstien, Philip, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. New York: Addison-Wesley, 1987.

Booch, Grady, Jim Rumbaugh, and Ivar Jacobson, *The UML Reference Manual*. New York: Addison-Wesley 1999.

Booch, Grady, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. New York: Addison-Wesley, 2000.

Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. New York: John Wiley & Sons, 1996.

Caffall, Dale Scott. and James Bret Michael. "A New Paradigm for Requirements Specification and Analysis of System-of-Systems." Wirsing, M., Balsamo, S., and Knapp, A., eds. *Lecture Notes in Computer Science: Proc. Monterey Workshop 2002: Radical Innovations of Software and Systems Engin. in the Future*. Berlin: Springer-Verlag, 2003. (Also appeared in Technical Report CS-2002-10, Dipartimento di Informatica, Università Cà Foscari di Venezia, Venezia, Italy, September 2002.)

Caffall, Dale Scott. "Conceptual Framework Approach For Systems-Of-Systems Software Developments." M.S. Thesis, Naval Postgraduate School, March 2003.

Committee for Naval Forces' Capability for Theater Missile Defense, Naval Studies Board, National Research Council. *Naval Forces' Capability for Theater Missile Defense*. Washington D.C.: National Academy Press, 2001.

Coulouris, George, Jean Dollimore, and Tim Kindberg. *Distributed Systems Concepts and Design*, 3rd Ed. New York: Addison-Wesley, 2001.

Gomaa, Hassan. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. New York: Addison-Wesley, 2001.

Greaney, Kevin J. "Evolving A Simulation Model Product Line Software Architecture From Heterogeneous Model Representations." Ph.D. Dissertation, Naval Postgraduate School, September 2003.

Kulak, Daryl, and Eamonn Guiney. *Use Cases Requirements in Context*. New York: ACM Press, 2001.

Leffingwell, Dean, and Don Widrig. *Managing Software Requirements A Unified Approach*. New York: Addison-Wesley, 2001.

Michael, James Bret, Phillip Pace, Man-Tak Shing, Murali Tummala and others, eds., "Test and Evaluation of the Ballistic Missile Defense System FY 03 Progress Report", Naval Postgraduate School, September 2003.

Missile Defense Agency. *Lexicon for Sensor Netting Applications*. Washington D.C. February 2003.

Missile Defense Agency. *MDA Glossary Version 4*. Washington D.C. November 2003.
<http://www.acq.osd.mil/bmdo/bmdolink/html/bmdolink.html>,
Accessed November 2003

OMNet++ Community Site. www.omnetpp.org, Accessed December 2003.

Rumbaugh, James, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. New York: Addison-Wesley, 1999.

Selic, Bran, and Jim Rumbaugh. *Using UML for Modeling Complex Real Time Systems*. April 1998

The Standish Group. *Charting the Seas of Information Technology*. The Standish Group International. 1994

www.whitehouse.gov, Accessed November 2003.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
Ft. Belvoir, VA
2. Dudley Knox Library, Code 52
Naval Postgraduate School
Monterey, CA
3. Dr. James Bret Michael
Computer Science Department
Naval Postgraduate School
Monterey, CA
4. Dr. Man Tak Shing
Computer Science Department
Naval Postgraduate School
Monterey, CA
5. Dr. Dan Boger
Naval Postgraduate School
Monterey, CA
6. Dr. Peter Denning
Computer Science Department
Naval Postgraduate School
Monterey, CA
7. Dr. Phil Pace
Electrical and Computer Engineering Department
Naval Postgraduate School
Monterey, CA
8. Dr. Murali Tummala
Electrical and Computer Engineering Department
Naval Postgraduate School
Monterey, CA
9. Mr. Dale Scott Caffall
Missile Defense Agency
Washington, DC